AFRL-AFOSR-VA-TR-2016-0015

# FAST IMPLICIT METHODS FOR ELLIPTIC MOVING INTERFACE PROBLEMS

John Strain
REGENTS OF THE UNIVERSITY OF CALIFORNIA THE

12/11/2015
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTA2
Arlington, Virginia 22203
Air Force Materiel Command

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*

| | | |
|---|---|---|
| **1. REPORT DATE** *(DD-MM-YYYY)*<br>14-12-2015 | **2. REPORT TYPE**<br>Final Performance | **3. DATES COVERED** *(From - To)*<br>30-09-2013 to 30-09-2015 |

| | |
|---|---|
| **4. TITLE AND SUBTITLE**<br>FAST IMPLICIT METHODS FOR ELLIPTIC MOVING INTERFACE PROBLEMS | **5a. CONTRACT NUMBER** |
| | **5b. GRANT NUMBER**<br>FA9550-11-1-0242 |
| | **5c. PROGRAM ELEMENT NUMBER**<br>61102F |
| **6. AUTHOR(S)**<br>John Strain | **5d. PROJECT NUMBER** |
| | **5e. TASK NUMBER** |
| | **5f. WORK UNIT NUMBER** |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**<br>REGENTS OF THE UNIVERSITY OF CALIFORNIA THE<br>2150 SHATTUCK AVE RM 313<br>BERKELEY, CA 94704 US | **8. PERFORMING ORGANIZATION REPORT NUMBER** |
| **9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**<br>AF Office of Scientific Research<br>875 N. Randolph St. Room 3112<br>Arlington, VA 22203 | **10. SPONSOR/MONITOR'S ACRONYM(S)**<br>AFRL/AFOSR RTA2 |
| | **11. SPONSOR/MONITOR'S REPORT NUMBER(S)** |

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
A DISTRIBUTION UNLIMITED: PB Public Release

**13. SUPPLEMENTARY NOTES**

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

**14. ABSTRACT**

Two notable advances in numerical methods were supported by this grant.
First, a fast algorithm was derived, analyzed, and tested for the Fourier transform of piecewise polynomials given on d-dimensional simplices in D-dimensional Euclidean space. These transforms play a key role in computational problems ranging from medical imaging to partial differential equations, and existing algorithms are inaccurate and/or prohibitively slow for d > 0. The algorithm employs low-rank approximation by Taylor series organized in a butterfly scheme, with moments evaluated by a new dimensional recurrence and simplex quadrature rules. For moderate accuracy and problem size it runs orders of magnitude faster than direct evaluation, and one to three orders of magnitude slower than the classical uniform Fast Fourier Transform.
Second, bilinear quadratures ---which numerically evaluate continuous bilinear maps, such as the L2 inner product, on continuous f and g belonging to known finite-dimensional function spaces---were analyzed and developed.

**15. SUBJECT TERMS**

elliptic moving interface

| **16. SECURITY CLASSIFICATION OF:** | | | **17. LIMITATION OF ABSTRACT** | **18. NUMBER OF PAGES** | **19a. NAME OF RESPONSIBLE PERSON** |
|---|---|---|---|---|---|
| **a. REPORT** | **b. ABSTRACT** | **c. THIS PAGE** | | | John Strain |
| Unclassified | Unclassified | Unclassified | UU | | **19b. TELEPHONE NUMBER** *(Include area code)* 510-642-3656 |

Two notable advances in numerical methods were supported by this grant.

First, a fast algorithm was derived, analyzed, and tested for the Fourier transform of piecewise polynomials given on d-dimensional simplices in D-dimensional Euclidean space. These transforms play a key role in computational problems ranging from medical imaging to partial differential equations, and existing algorithms are inaccurate and/or prohibitively slow for d > 0. The algorithm employs low-rank approximation by Taylor series organized in a butterfly scheme, with moments evaluated by a new dimensional recurrence and simplex quadrature rules. For moderate accuracy and problem size it runs orders of magnitude faster than direct evaluation, and one to three orders of magnitude slower than the classical uniform Fast Fourier Transform.

Second, bilinear quadratures ---which numerically evaluate continuous bilinear maps, such as the L2 inner product, on continuous f and g belonging to known finite-dimensional function spaces---were analyzed and developed. Such maps arise in Galerkin methods for differential and integral equations. Bilinear quadratures were constructed over arbitrary D-dimensional domains. In one dimension, integration rules of this type include Gaussian quadrature for polynomials and the trapezoidal rule for trigonometric polynomials. A numerical procedure for constructing bilinear quadratures was developed and validated.

# FAST IMPLICIT METHODS FOR ELLIPTIC MOVING INTERFACE PROBLEMS

John Strain

*Department of Mathematics, University of California, 970 Evans Hall #3840, Berkeley, California 94720-3840*

**Abstract**

A fast algorithm is derived, analyzed, and tested for the Fourier transform of piecewise polynomials given on $d$-dimensional simplices in $D$-dimensional Euclidean space. These transforms play a key role in computational problems ranging from medical imaging to partial differential equations, and existing algorithms are inaccurate and/or prohibitively slow for $d > 0$. The algorithm employs low-rank approximation by Taylor series organized in a butterfly scheme, with moments evaluated by a new dimensional recurrence and simplex quadrature rules. For moderate accuracy and problem size it runs orders of magnitude faster than direct evaluation, and one to three orders of magnitude slower than the classical uniform Fast Fourier Transform.

*Key words:* Fourier series, butterfly algorithm, spectral methods, numerical methods,
*PACS:* 02.30.Jr, 02.60.-x, 02.30.Nw, 46.15.-x, 47.11.-j, 02.70.-c

*Email address:* `strain@math.berkeley.edu` (John Strain).
*URL:* `http://math.berkeley.edu/~strain` (John Strain).

## 1 Introduction

In this paper, we present a new algorithm for the fast evaluation of the exact Fourier transform

$$\widehat{f}(t_k) = \sum_{j=1}^{N} \int_{S_j} \exp(\mathrm{i} t_k^T s) f_j(s)\, \mathrm{d}s, \qquad 1 \le k \le N, \tag{1}$$

of piecewise-polynomial densities $f_j(s)$ defined on simplices $S_j \subset \mathbf{R}^D$, at arbitrary points $t_k \in \mathbf{R}^D$. Here $\mathrm{i} = \sqrt{-1}$, the ambient dimension $D \ge 1$, the simplex dimension $d$ satisfies $D \ge d \ge 0$, and each $S_j$ is a $d$-dimensional simplex of the form

$$S = \{s = v_0 + \sum_{i=1}^{d} \theta_i (v_i - v_0) \mid \theta_i \ge 0, \sum_{i=1}^{d} \theta_i \le 1\}, \tag{2}$$

consisting of all convex combinations of $d+1$ given vertices $v_i \in \mathbf{R}^D$. A wide variety of useful special cases occur frequently in applications.

When the simplex dimension $d = 0$, each simplex $S_j$ is a point $s_j$, and each $f_j$ is a constant. Transform (1) becomes the pointwise nonuniform Fourier transform

$$\widehat{f}(t_k) = \sum_{j=1}^{N} \exp(\mathrm{i} t_k^T s_j) f_j, \qquad 1 \le k \le N, \tag{3}$$

where $s_j \in \mathbf{R}^D$ and $t_k \in \mathbf{R}^D$ are given $D$-vectors and $f_j$ is a complex number. The special case where $s_j$ and $t_k$ form regular cubical grids corresponds to the classical uniform Fast Fourier Transform [1,2]. Several classes of fast algorithms for pointwise nonuniform transforms such as (3) are now well-known [3–8], but fast algorithms for the more general transform (1) are less developed [9–13].

When the simplex dimension $d = 1$, each simplex $S_j$ is a line segment $[u_j, v_j]$ connecting endpoints $u_j$, $v_j \in \mathbf{R}^D$, and parametrized by $s = u_j + \sigma(v_j - u_j)$ for $0 \le \sigma \le 1$. Transform (1) becomes

$$\widehat{f}(t_k) = \sum_{j=1}^{N} \|v_j - u_j\| \int_{0}^{1} \exp(\mathrm{i} t_k^T (u_j + \sigma(v_j - u_j))) f_j(\sigma)\, \mathrm{d}\sigma, \qquad 1 \le k \le N.$$

When $d = 2$, each simplex $S_j$ is a triangle $\Delta(u_j, v_j, w_j)$ with vertices $u_j$, $v_j$, $w_j \in \mathbf{R}^D$, parametrized by $s = u_j + \sigma(v_j - u_j) + \theta(w_j - u_j)$ where $0 \le \theta \le 1$,

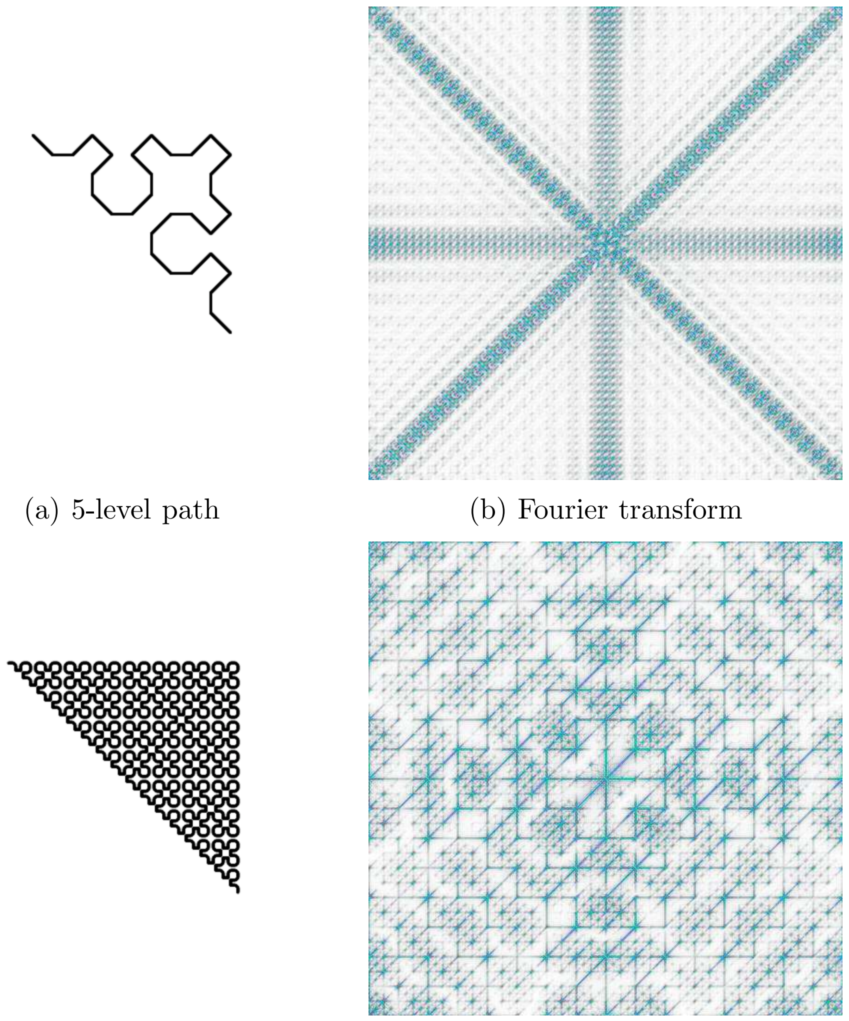
$0 \leq \sigma \leq 1$, $\sigma + \theta \leq 1$. Transform (1) becomes

$$\widehat{f}(t_k) = \sum_{j=1}^{N} \sqrt{\|v_j - u_j\|^2 \|w_j - u_j\|^2 - ((v_j - u_j)^T (w_j - u_j))^2}$$
$$\cdot \int_0^1 \int_0^{1-\sigma} \exp(\mathrm{i} t_k^T (u_j + \sigma(v_j - u_j) + \theta(w_j - u_j))) f_j(\sigma, \theta) \, \mathrm{d}\theta \, \mathrm{d}\sigma, \qquad 1 \leq k \leq N.$$

For simplex dimension $d \geq 1$, the transforms (1) arise in the discretization of geometric objects by line segments, triangles and tetrahedra. For example, Fig. 1 displays $L$-level approximate Sierpinski paths composed of line segments $S_j$ with simplex dimension $d = 1$ in ambient dimension $D = 2$, together with the Fourier transforms of unit data $f = 1$ on these paths. As $L$ increases, the Sierpinski paths tend to fill a triangle $T$ and the Fourier transform $\widehat{f}$ tends to the Fourier transform of the characteristic function of $T$.

Our algorithm evaluates transform (1) in $O(N \log N \log \epsilon)$ work to accuracy $\epsilon$, for arbitrary $d$ and $D$. It is globally structured as in the butterfly algorithm of [14], with local transformations based on multidimensional Taylor series. Thus it groups source simplices $S_j$ and target points $t_k$ into hierarchical tree structures, approximates the kernel $\exp(\mathrm{i}t^T s)$ by low-rank expansions, and transforms the low-rank expansions from source-local to target-local form. The main new ingredient is a stable efficient dimensional recurrence for local source moments of polynomials over simplices. Our derivation, analysis and implementation all operate with arbitrary ambient dimension $D \geq 1$, arbitrary simplex dimension $0 \leq d \leq D$, and arbitrary polynomial degree $\deg(f_j) = p \geq 0$. Despite this generality, our implementation runs orders of magnitude faster than direct evaluation and even compares favorably with the classical uniform Fast Fourier Transform.

The paper is organized as follows. In Section 2, we review mathematical tools such as error bounds for Taylor expansion of exponential functions, and spatial tree structures for localizing target points. In Section 3, we combine these tools to derive a pointwise ($d = 0$) butterfly algorithm similar to [14]. In Section 4, we generalize the butterfly algorithm to piecewise polynomials $f_j$ on simplices of dimension $d \geq 1$. The main new tool is a stable efficient dimensional recurrence which evaluates exponential-polynomial moments in dimension $d \geq 1$ by a combination of simplex quadrature and recurrence to simplex dimension $d - 1$. In Section 5 we present numerical experiments which verify the efficiency and accuracy of the approach. In Section 6 we discuss potential speedups with optimized basis functions [15], the evaluation of Fourier-Galerkin matrix elements [16], and extensions to Laplace and Gauss transforms [17–19].



(a) 5-level path (b) Fourier transform

(c) 10-level path (d) Fourier transform

Fig. 1. Sierpinski paths and Fourier transforms

## 2 Mathematical preliminaries

We review classical tools for low-rank kernel approximation and hierarchical point clustering. The exponential kernel of transform (1) is approximated by Taylor expansion. A simple error estimate delineates the region where Taylor expansion is accurate. Tree structures are developed to organize source and target points into groups where Taylor expansion is accurate. The classical pointwise butterfly algorithm combines these tools, while our new piecewise-polynomial butterfly algorithm involves additional ingredients.

### *2.1 Low-rank kernel approximation*

The multidimensional Taylor expansion

$$\exp(\mathrm{i}t^T s) = \sum_{i=0}^{\infty} \frac{\mathrm{i}^n \left(\sum_{j=1}^{D} t_j s_j\right)^n}{n!} = \sum_{\alpha \geq 0} \frac{\mathrm{i}^{|\alpha|}}{\alpha!} t^\alpha s^\alpha \tag{4}$$

of the complex exponential kernel follows immediately from the multinomial theorem

$$\left(\sum_{j=1}^{D} t_j\right)^n = n! \sum_{|\alpha|=n} \frac{t^\alpha}{\alpha!}. \tag{5}$$

Here

$t$ and $s$ are real or complex $D$-vectors,
$t^T s = t_1 s_1 + \cdots + t_D s_D$ is their inner product,
$\alpha = (\alpha_1, \ldots, \alpha_D) \in N^D$ is a $D$-dimensional multiindex of order
$|\alpha| = \alpha_1 + \cdots + \alpha_D$,
$t^\alpha = t_1^{\alpha_1} \cdots t_D^{\alpha_D}$ and $s^\alpha$ are monomials, and
$\alpha! = \alpha_1! \alpha_2! \cdots \alpha_D!$.

According to Stirling's inequality $m! \geq (m/e)^m$, terminating expansion (4) after $M = \binom{m+D}{D}$ terms of order $|\alpha| \leq m$ incurs truncation error $E_m$ bounded by

$$|E_m| = \left| \sum_{|\alpha|>m} \frac{1}{\alpha!} t^\alpha s^\alpha \right| = \left| \sum_{k=m+1}^{\infty} \frac{\mathrm{i}^k}{k!} (t^T s)^k \right| \leq \sum_{k=m+1}^{\infty} \frac{R^k}{k!} \leq \left( \frac{R\mathrm{e}}{m} \right)^m \tag{6}$$

as long as $|t^T s| \le R$ where $R/m \le 0.27$ or equivalently $m \ge 3.8R$. For example, with $R = 1$ accuracy $\epsilon = 10^{-6}$ is guaranteed with $m = 11$ and accuracy $\epsilon = 10^{-12}$ is guaranteed with $m = 16$. In general, the number of terms $M$ increases polylogarithmically as $\epsilon$ decreases.

The utility of expansion (6) can be easily demonstrated with unrealistically distributed sets of targets and sources. Suppose all the sources $s_j$ and targets $t_k$ in the pointwise transform (3) are clustered near 0 so that $|t_k^T s_j| \le R$, where $R$ and $m$ are chosen to guarantee error $|E_m| \le \epsilon$. Then expansion (4) speeds up the pointwise transform (3) as follows:

$$\begin{aligned}\widehat{f}(t_k) &= \sum_{j=1}^{N} \exp(\mathrm{i} t_k^T s_j) f_j \\ &= \sum_{j=1}^{N} \sum_{|\alpha| \le m} \frac{\mathrm{i}^{|\alpha|}}{\alpha!} t_k^\alpha s_j^\alpha f_j + F_m \\ &= \sum_{|\alpha| \le m} C_\alpha t_k^\alpha + F_m.\end{aligned} \tag{7}$$

Here the coefficients $C_\alpha$ defined by

$$C_\alpha = \frac{\mathrm{i}^{|\alpha|}}{\alpha!} \sum_{j=1}^{N} s_j^\alpha f_j$$

encode the sources $s_j$ and strengths $f_j$ into $M$ moments, and $|F_m| \le \epsilon \sum |f_j|$ bounds the error. All $M$ moments up to order $m$ can be evaluated in $O(MN)$ work, while the resulting polynomial approximation to $\widehat{f}(t)$ can be evaluated at $N$ targets $t_k$ in $O(MN)$ work. Thus the rank-$M$ kernel approximation

$$\exp \mathrm{i} t^T s = \sum_{|\alpha| \le m} \frac{\mathrm{i}^{|\alpha|}}{\alpha!} t_k^\alpha s_j^\alpha + E_m \tag{8}$$

gives an $O(N)$ algorithm with a constant factor $O(M)$ depending polylogarithmically on the accuracy $\epsilon$.

The sources $s_j$ and targets $t_k$ are not conveniently clustered in most applications. Indeed, in the classical one-dimensional Fast Fourier Transform [1,2], $s_j = 2\pi j$ and $t_k = k/N$ for $1 \le j, k \le N$, so $0 \le t_k^T s_j \le 2\pi N \to \infty$ as $N \to \infty$. Thus we employ a collection of expansions (6) centered about arbitrary target centers $\tau$ and source centers $\sigma$. Each expansion represents the transform due to sources $s_j$ in a cubical cell $S$ centered at $s = \sigma$, evaluated at targets $t_k$ in a cell $T$ centered at $t = \tau$:

$$\begin{aligned}\sum_{s_j\in S}\exp(\mathrm{i}t_k^Ts_j)f_j &= \sum_{s_j\in S}\exp(\mathrm{i}\tau^T\sigma+\mathrm{i}\tau^T(s_j-\sigma)+\mathrm{i}(t_k-\tau)^T(s_j-\sigma)+\mathrm{i}(t_k-\tau)^T\sigma)f_j\\ &= \sum_{|\alpha|\le m}\exp(\mathrm{i}\tau^T\sigma)\frac{\mathrm{i}^{|\alpha|}}{\alpha!}\sum_{s_j\in S}(s_j-\sigma)^\alpha\exp(\mathrm{i}\tau^T(s_j-\sigma))f_j\\ &\quad\cdot(t_k-\tau)^\alpha\exp(\mathrm{i}(t_k-\tau)^T\sigma)+F_m\\ &= \sum_{|\alpha|\le m}C_\alpha(\sigma,\tau)(t_k-\tau)^\alpha\exp(\mathrm{i}(t_k-\tau)^T\sigma)+F_m\end{aligned}$$

where

$$C_\alpha(\sigma,\tau)=\exp(\mathrm{i}\tau^T\sigma)\frac{\mathrm{i}^{|\alpha|}}{\alpha!}\sum_{s_j\in S}(s_j-\sigma)^\alpha\exp(\mathrm{i}\tau^T(s_j-\sigma))f_j \tag{9}$$

and $|F_m|\le\epsilon\sum|f_j|$ if

$$|(t_k-\tau)^T(s_j-\sigma)|\le R \quad\text{where}\quad \left(\frac{Re}{m}\right)^m\le\epsilon \tag{10}$$

for all $t_k\in T$ and $s_j\in S$. Thus the transform due to sources in $S$, evaluated at targets in $T$, is approximated with a kernel

$$\exp(\mathrm{i}\tau^T\sigma)\sum_{|\alpha|\le m}(s_j-\sigma)^\alpha\exp(\mathrm{i}\tau^T(s_j-\sigma))\frac{\mathrm{i}^{|\alpha|}}{\alpha!}(t_k-\tau)^\alpha\exp(\mathrm{i}(t_k-\tau)^T\sigma)\tag{11}$$

of rank $M=\binom{m+D}{D}$, whenever inequalities (10) hold. The accuracy of this approximation is controlled by the geometry of $S$ and $T$, rather than the separation required by classical multipole methods [20,21]. In this paper, we control accuracy by controlling the relative sizes of $S$ and $T$, but it is also possible to divide the sources and targets into groups with selected orientations $\theta$ in $t^Ts=\|t\|\|s\|\cos\theta$.

Butterfly algorithms apply low-rank approximate kernels (11) to spatial clusters of sources and targets, via a pair of translation lemmas that shift the coefficients $C_\alpha(\sigma,\tau)$ of (9) to smaller target cells and larger source cells. These lemmas are proved by expanding the exponential in Taylor series and applying the binomial theorem respectively:

**Lemma 1** *Given sources $s_j\in S$ and targets $t_k\in T$, the coefficient vector $C(\sigma,\tau_1)\in\mathbf{C}^M$ relative to a smaller target cell $T_1\subset T$ with center $\tau_1$ is approximated by an upper triangular matrix multiply*

$$C_\alpha(\sigma,\tau_1)=\exp(\mathrm{i}(\tau_1-\tau)^T\sigma)\sum_{\beta\ge0}\binom{\beta+\alpha}{\beta}(\tau_1-\tau)^\beta C_{\beta+\alpha}(\sigma,\tau)$$

$$= \sum_{\alpha \leq \beta,\, |\beta| \leq m} R_{\alpha\beta}(\sigma, \tau - \tau_1) C_\beta(\sigma, \tau) + E \tag{12}$$

*where*

$$R_{\alpha\beta}(\sigma, \tau) = \exp(\mathrm{i}(\tau^T \sigma)) \binom{\beta}{\beta - \alpha} \tau^{\beta-\alpha} \quad \textit{for} \quad \beta \geq \alpha.$$

*If $|\alpha| \leq m$, the error $E$ incurred by truncating formula (12) after terms of order $|\beta| \leq m$ is bounded by $\rho^{-m}\epsilon \sum |f_j|$ whenever $|(\tau_1 - \tau)^T (s_j - \sigma)| \leq R/\rho$.*

The error estimate for truncating the infinite series follows immediately from inequality (10) since $T_1 \subset T$.

**Lemma 2** *Given coefficients $C(\sigma_1, \tau)$ for source cell $S$ and target cell $T_1$, the Taylor coefficients $C_\alpha(\sigma, \tau)$ relative to a larger source cell $S \supset S_1$ with center $\sigma$ are given by an exact lower triangular matrix multiply*

$$\begin{aligned} C_\alpha(\sigma, \tau) &= \exp(\mathrm{i}\tau^T(\sigma_1 - \sigma)) \sum_{\beta \leq \alpha} \frac{\mathrm{i}^{|\alpha-\beta|}}{(\alpha - \beta)!} (\sigma_1 - \sigma)^{\alpha-\beta} C_\beta(\sigma_1, \tau) \\ &= \sum_{\beta \leq \alpha} L_{\alpha\beta}(\sigma_1 - \sigma, \tau) C_\beta(\sigma_1, \tau). \end{aligned} \tag{13}$$

*where*

$$L_{\alpha\beta}(\sigma, \tau) = \exp(\mathrm{i}\tau^T \sigma) \frac{\mathrm{i}^{|\alpha-\beta|}}{(\alpha - \beta)!} \sigma^{\alpha-\beta}$$

*for $\alpha \leq \beta$.*

After using Lemma 2 to shift coefficient vectors from several different source subcells $S_1, \ldots, S_n$ to a common superset $S$ with center $\sigma$, the resulting expansions can be merged into a single expansion by adding the coefficients:

$$C_\beta(\sigma, \tau) = \sum_{j=1}^{n} \sum_{\beta \leq \alpha} L_{\alpha\beta}(\sigma_j - \sigma, \tau) C_\beta(\sigma_j, \tau).$$

In the butterfly algorithm, the matrices $L(\sigma - \sigma_j, \tau_i)$ and $R(\sigma_j, \tau - \tau_i)$ operate on $2^D$ expansions $C(\sigma_j, \tau)$ representing the combined effect of sources in the $2^D$ children $S_j$ of a source cell $S$, evaluated at targets in a target cell $T$. The result is another set of $2^D$ expansions $C(\sigma, \tau_i)$ representing sources in the parent source cell $S$ to targets in each of the $2^D$ children $T_i$ of $T$. The combined operation reduces source locality and increases target locality by a block matrix-vector multiply with $2^D$ $M \times M$ blocks:

$$C(\sigma, \tau_i) = \sum_{j=1}^{2^D} L(\sigma_j - \sigma, \tau_i) R(\sigma_j, \tau - \tau_i) C(\sigma_j, \tau). \tag{14}$$

The butterfly algorithm presented in this paper factorizes the transform (1) by repeated application of Eq. (14) to moments formed with groups of simplices and densities. The factorization process is abstracted and generalized in [22].

## *2.2 Hierarchical point clustering*

The expansion shifting and merging lemmas of subsection 2.1 operate within a data structure which organizes source and target objects $S_j$ and $t_k$ into cells $S$ and $T$ so that inequality (10) is locally satisfied. Collections of geometric objects such as the source simplices and target points in Eq. (1) can be efficiently organized into local groups by a variety of tree-based data structures [23]. We employ a $2^D$-ary tree which generalizes quadtrees and octrees to arbitrary space dimension $D$. In the present pointwise case where $d = 0$, its construction is straightforward.

Suppose $N$ points $s_j \in \mathbf{R}^D$ are all contained in a hypercubical unit cell $S_{0,0} = [\sigma_1 - R, \sigma_1 + R] \times \ldots \times [\sigma_D - R, \sigma_D + R] =: \sigma_{0,0} + R\mathcal{Q}$ with center $\sigma_{0,0}$ and width $2R$. Here $\mathcal{Q}$ denotes the unit cell $[-1, 1]^D$ and $+$ denotes addition of sets. We can build a $L$-level tree structure $\mathcal{S}_L$ from the root downward as follows. Let the root cell be $S_{0,0}$, comprising the root level 0 of $\mathcal{S}$. For level $l = 0$ to $L - 1$, for $J = 0$ to $2^{Dl} - 1$, divide each cubical cell $S_{l,J} = \sigma_{l,J} + 2^{-l} R\mathcal{Q}$ into $2^D$ cubical children $S_{l+1,2^D J+j} = \sigma_{l+1,2^D J+j} + 2^{-l-1} R\mathcal{Q}$ for $j = 0$ to $2^D - 1$, and record them on level $l + 1$. As each child cell is constructed, construct pointers from the cell to points $s_k$ located inside it and vice versa. The resulting structure $\mathcal{S}_L$ can be used to answer many range-type queries such as finding nearest neighbors. Its $O(NL)$ cost is rarely optimal for any specific task, but its flexibility and efficiency has made it ubiquitous in computational science. Fig. 2 illustrates the construction.

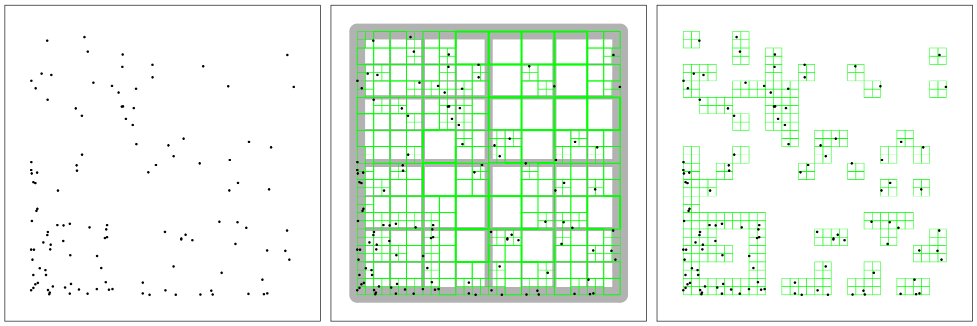

(a) Points (b) Tree (c) Cells

Fig. 2. Points in two dimensions, a 6-level tree, and nonempty child cells.

## 3 A pointwise butterfly algorithm

A butterfly algorithm for the pointwise transform (3) can be based on low-rank expansion and hierarchical point clustering [14,15,24–26]. It localizes sources and targets into tree structures, forms source-local expansion coefficients, shifts and merges them systematically, and evaluates target-local expansions. We derive these pointwise algorithms in four steps before generalizing to the piecewise-polynomial transform (1) in Section 4.

First, we build spatial tree structures $\mathcal{S}_L$ and $\mathcal{T}_L$ for the source and target points. These structures cover the data regions $S$ and $T$ by superimposed layers $l = 0$ (the root) to $L$ (the finest level). The number $L+1$ of levels required is determined by $|S||T|$ where $|S|$ and $|T|$ are the diameters of the source and target point collections respectively. Layer $l$ consists of $2^{Dl}$ rectangular cells $S_{l,j}$ or $T_{l,j}$ numbered from 0 to $2^{Dl}-1$, whose union partitions the points $s_k$ or $t_k$. The error in an $M$-term Taylor expansion (4) of the exponential kernel, for sources located in a level-$l$ cell of $\mathcal{S}_L$ and targets located in a level-$(L-l)$ cell of $\mathcal{T}_L$ will be bounded by $\epsilon \sum |f_j|$ for all $l$ if the inequalities

$$2^{-L}|S||T| \leq R \qquad \text{and} \qquad \left(\frac{R\mathrm{e}}{m}\right)^m \leq \epsilon \tag{15}$$

are satisfied. Typically $2^{DL} = O(N)$ so that each level-$L$ cell contains on average $O(1)$ points. However, uniform distribution of the points is not assumed and does not alter the cost of the algorithm.

Second, we create source-local coefficient vectors $C(\sigma_{L,j}, \tau_{0,0})$ which represent the effect of sources $s_k$ in each leaf cell $S_{L,j}$ with center $\sigma_{L,j}$ on the finest level $L$ of the source tree $\mathcal{S}_L$, on all targets $t_k$ in the root cell $T_{0,0}$ with center $\tau_{0,0}$ on level 0 of the target tree $\mathcal{T}_L$. Thus

$$C_\alpha(\sigma_{L,j}, \tau_{0,0}) = \exp(\mathrm{i}\tau_{0,0}^T \sigma_{L,j}) \frac{\mathrm{i}^{|\alpha|}}{\alpha!} \sum_{s_k \in S_{L,j}} (s_k - \sigma_{L,j})^\alpha \exp(\mathrm{i}\tau_{0,0}^T (s_k - \sigma_{L,j})) f_k.$$

The total cost of coefficient evaluation is $O(NM)$ since each source contributes to $M$ coefficients in a single leaf cell per source. Once these coefficients have been evaluated, the field at each target point $t_k$ could be accurately approximated by summing the source-local expansions:

$$\widehat{f}(t_k) = \sum_{j=0}^{2^{DL}-1} \exp(\mathrm{i}(t_k - \tau_{0,0})^T \sigma_{L,j}) \sum_{|\alpha| \leq m} C_\alpha(\sigma_{L,j}, \tau_{0,0})(t_k - \tau_{0,0})^\alpha.$$

However, the $O(N^2)$ cost of this computation would be prohibitive since $2^{DL} =$

$O(N)$. An efficient evaluation scheme requires target-local coefficient vectors $C(\sigma_{0,0}, \tau_{L,j})$ which represent the effect of all sources to each leaf cell of the target tree $\mathcal{T}_L$, and evaluates one expansion

$$\widehat{f}(t_k) = \exp(\mathrm{i}(t_k - \tau_{L,j})^T \sigma_{0,0}) \sum_{|\alpha| \leq m} C_\alpha(\sigma_{0,0}, \tau_{L,j})(t_k - \tau_{L,j})^\alpha,$$

at each target $t_k$ in $T_{L,j}$. The total cost of evaluating these expansions is $O(NM)$ since each target evaluation involves $M$ coefficients.

The third step of the butterfly algorithm computes the target-local coefficient vectors $C(\sigma_{0,0}, \tau_{L,j})$ which approximate the total field, at targets $t_k$ in the finest level-$L$ target cells $T_{L,j}$ with centers $\tau_{L,j}$, due to all the sources $s_k$ in the source root cell $S_{0,0}$ with center $\sigma_{0,0}$. The algorithm proceeds by doubling source cells and halving target cells, and therefore requires $L = O(\log N)$ substeps. The substep from source level $l$ and target level $L - l$ to source parent level $l - 1$ and target child level $L - l + 1$, consists of two operations.

In the first operation, on source level $l$ and target level $(L - l)$, for $J = 0$ to $2^{D(L-l)} - 1$, each target cell $T_{L-l,J}$ is *split* into $2^D$ children $T_{L-l+1,2^D J+j} \subset T_{L-l,J}$ with $j = 0$ to $2^D - 1$. Each coefficient vector $C(\sigma_{l,i}, \tau_{L-l,J})$ relative to the parent center $\tau_{L-l,J}$ is converted to another coefficient vector

$$C(\sigma_{l,i}, \tau_{L-l+1,2^D J+j}) = R(\sigma_{l,i}, \tau_{L-l,J} - \tau_{L-l+1,2^D J+j})C(\sigma_{l,i}, \tau_{L-l,J})$$

relative to each child center $\tau_{L-l+1,2^D J+j}$, using the $M \times M$ matrices $R(\sigma_{l,i}, \tau_{L-l,J} - \tau_{L-l+1,2^D J+j})$ defined in Eq. (12).

In the second operation, each group of $2^D$ level-$l$ source cell siblings $S_{l,i} \subset S_{l-1,I}$ with $I = \lfloor i/2^D \rfloor$, is *merged* into their parent $S_{l-1,I}$. For $J = 0$ to $2^{D(L-l+1)} - 1$, each coefficient vector $C(\sigma_{l,i}, \tau_{L-l+1,2^D J+j})$ relative to source child center $\sigma_{l,i}$ is converted to a coefficient vector

$$B^i(\sigma_{l-1,I}, \tau_{L-l+1,2^D J+j}) = L(\sigma_{l,i} - \sigma_{l-1,I}, \tau_{L-l+1,2^D J+j})C(\sigma_{l,i}, \tau_{L-l+1,J})$$

relative to the source parent center $\sigma_{l-1,I}$, using the $M \times M$ matrices $L(\sigma_{l,i} - \sigma_{l-1,I}, \tau_{L-l+1,2^D J+j})$ defined in Eq. (13). Since the $2^D$ resulting coefficient vectors $B^i$ have the same source center $\sigma = \sigma_{l-1,I})$, and the same target center $\tau_1 = \tau_{L-l+1,2^D J+j}$, they are summed coefficient by coefficient:

$$C(\sigma, \tau_1) = \sum_{i=1}^{2^D} B^i(\sigma_{l,i}, \tau_1)$$

$$= \sum_{i=1}^{2^D} L(\sigma_{l,i} - \sigma, \tau_1) R(\sigma_{l,i}, \tau_{L-l,J} - \tau_1) C(\sigma_{l,i}, \tau_{L-l,J}).$$

After these two substeps, target cells have been halved and source cells have been doubled, so the error control estimate (10) is preserved. Arranging the fan-out matrices $R$ and fan-in matrices $L$ symmetrically as in Fig. 3 illustrates the butterfly algorithm [**?**,**?**,**?**].

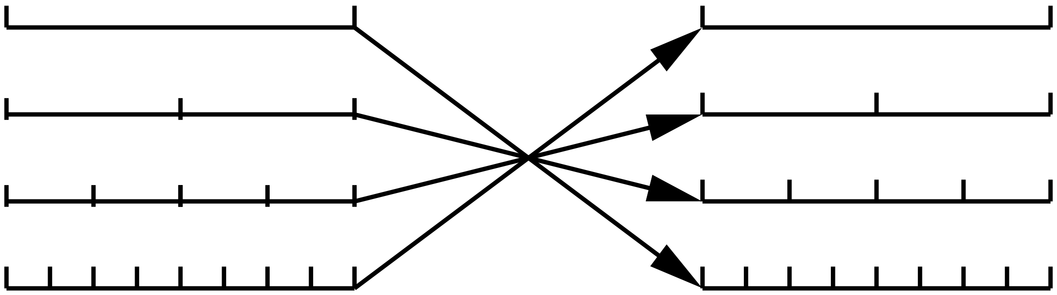


Fig. 3. Interactions between opposite levels of the source and target trees give the butterfly algorithm its name.

After $L$ steps of the butterfly scheme, a coefficient vector $C(\sigma_{0,0}, \tau_{L,j})$ has been computed for each target cell $T_{L,j}$ on the finest level $L$, and each coefficients vector represents the field due to all the sources, evaluated at any $t_k \in T_{L,j}$:

$$\widehat{f}(t_k) = \exp\left(\mathrm{i}(t_k - \tau_{L,j})^T \sigma_{0,0}\right) \sum_{|\alpha| \le m} C_\alpha(\sigma_{0,0}, \tau_{L,j})\,(t_k - \tau_{L,j})^\alpha \tag{16}$$

Hence the transform can be accurately evaluated at each target $t_k$ in $O(MN)$ time, where $M$ depends polylogarithmically on the accuracy required. A detailed algorithm is exhibited in Fig. 4.

Fig. 4. A pointwise butterfly algorithm

**Step 1 - Localization**

Sort sources $s_j$ and targets $t_k$ into leaf cells of $L$-level trees $\mathcal{S}_L$ and $\mathcal{T}_L$

**Step 2 - Compute source-centered coefficients**

```
for $j = 0 \ldots 2^{DL} - 1$
    $S = S_{L,j}$
    $\sigma = \sigma_{L,j}$
    $\tau = \tau_{0,0}$
    for $|\alpha| \leq m$
        $C_\alpha(\sigma,\tau) = \exp(\mathrm{i}\tau^T\sigma)\frac{\mathrm{i}^{|\alpha|}}{\alpha!}\sum_{s_k \in S}(s_k - \sigma)^\alpha \exp(\mathrm{i}\tau^T(s_k - \sigma))f_k$
    end for
end for
```

**Step 3 - Butterfly**

```
for $l = L \ldots 1$
    for $I = 0 \ldots 2^{Dl} - 1$
        $\sigma = \sigma_{l-1,I}$
        for $J = 0 \ldots 2^{D(L-l)} - 1$
            $\tau = \tau_{L-l,J}$
            for $j = 0 \ldots 2^D - 1$
                $\tau_1 = \tau_{L-l+1,2^D J+j}$
                $C(\sigma,\tau_1) = \sum_{i=0}^{2^D-1} L(\sigma_{l,I/2^D+i} - \sigma, \tau_1)R(\sigma_{l,I/2^D+i}, \tau - \tau_1)C(\sigma_{l,I/2^D+i}, \tau)$
            end for
        end for
    end for
end for
```

**Step 4 - Evaluate target-centered expansions**

```
for $j = 0 \ldots 2^{DL} - 1$
    $\sigma = \sigma_{0,0}$
    $T = T_{L,j}$
    $\tau = \tau_{L,j}$
    for $t_k \in T_{L,j}$
        $\widehat{f}(t_k) = \exp(\mathrm{i}(t_k - \tau)^T\sigma))\sum_{|\alpha| \leq m} C_\alpha(\sigma,\tau)(t_k - \tau)^\alpha$
    end for
end for
```

## 4 Piecewise-polynomial distributions

Given $N$ polynomial densities $f_j$ on $d$-dimensional simplices $S_j \subset \mathbf{R}^D$, we derive a fast algorithm for evaluating the Fourier transform

$$\widehat{f}(t_k) = \sum_{j=1}^{N} \int_{S_j} \exp(\mathrm{i} t_k^T s) f_j(s)\,\mathrm{d}s \tag{17}$$

at $N$ points $t_k \in \mathbf{R}^D$. Our algorithm combines the low-rank kernel approximation, tree structures and butterfly scheme of Sections 2 and 3 with a new dimensional recurrence for evaluating exponential-polynomial moments (Section 4.1.1) and a flexible tree structure for localizing simplices (Section 4.1.2).

Suppose a given simplex $S_j$ is contained in a cubical cell $S = \sigma + R_S \mathcal{Q}$ and $t_k$ lies in a cubical cell $T = \tau + R_T \mathcal{Q}$ where $R = R_S R_T$ satisfies

$$\left(\frac{R\mathrm{e}}{m}\right)^m \leq \epsilon. \tag{18}$$

Then

$$\exp(\mathrm{i} t_k^T s) = \exp(\mathrm{i}\tau^T \sigma) \sum_{|\alpha| \leq m} \frac{\mathrm{i}^{|\alpha|}}{\alpha!} (s-\sigma)^\alpha \exp(\mathrm{i}\tau^T (s-\sigma))\,(t_k - \tau)^\alpha \exp(\mathrm{i}(t_k-\tau)^T \sigma) + E_m$$

where $|E_m| \leq \epsilon$. Hence integrating over $S_j$ gives

$$\int_{S_j} \exp(\mathrm{i} t_k^T s) f_j(s)\,\mathrm{d}s = \sum_{|\alpha| \leq m} C_\alpha(\sigma,\tau)\,(t_k-\tau)^\alpha \exp(\mathrm{i}(t_k-\tau)^T\sigma) + F_m \tag{19}$$

where $|F_m| \leq \epsilon \int |f_j|$ and

$$\begin{aligned} C_\alpha(\sigma,\tau) &= \exp(\mathrm{i}\tau^T\sigma)\frac{\mathrm{i}^{|\alpha|}}{\alpha!} \int_{S_j} (s-\sigma)^\alpha \exp(\mathrm{i}\tau^T(s-\sigma)) f_j(s)\,\mathrm{d}s \\ &= \exp(\mathrm{i}\tau^T\sigma)\frac{\mathrm{i}^{|\alpha|}}{\alpha!} F_\alpha(d, S_j, f_j). \end{aligned}$$

Here the $M$-vector $F$ of exponential-polynomial moments is defined by

$$F_\alpha(d,S,f) = \int_S (s-\sigma)^\alpha \exp(\mathrm{i}\tau^T(s-\sigma)) f(s)\,\mathrm{d}s \tag{20}$$

for a polynomial density $f$ on a simplex $S$. The source center $\sigma$, target center $\tau$, and ambient dimension $D$ are omitted from the notation for simplicity. In Section 4.1.1, we derive efficient methods for evaluating $F$.

## *4.1 Exponential-polynomial moments*

Fourier transforms of piecewise-polynomial data over simplices involve exponential-polynomial moments defined by

$$F_\alpha(d, S, f) = \int_S \exp(\mathrm{i} t^T s)(s - \sigma)^\alpha f(s)\, \mathrm{d}s, \tag{21}$$

computed for all multiindices $\alpha$ with $|\alpha| \leq m$ and a given degree-$p$ polynomial $f$ defined on the $d$-dimensional simplex $S$. The target point $t$, ambient dimension $D$ and cell center $\sigma$ are omitted from notation for simplicity.

For example, direct evaluation of transform (1) without the butterfly algorithm is equivalent to the summation of $N$ moments $F(d, S_j, f_j, 0)$ of order $|\alpha| = 0$ for each point of evaluation $t_k$. Expansion (19) requires moments $F(d, S_j, f_j, \alpha)$ for all orders up to $|\alpha| = m$, with $t = \tau$ ranging over various target cell centers. Similar moments formed with polynomial, exponential or bandlimited bases occur in other butterfly algorithms for pointwise data with $d = 0$. When $d > 0$, no algorithm is known for their exact evaluation, and the rapid oscillation of the exponential kernel renders them resistant to standard numerical quadratures.

We evaluate these moments by a recursively branching three-step procedure. First, we extract the variation of the exponential perpendicular to $S$ (subsection 4.1.1). Second, if the remaining parallel variation is small, we evaluate the moment directly by numerical quadrature (Section 4.1.2). Third, we reduce the simplex dimension $d$ by dimensional recurrence (Section 4.1.3), continuing recursively until $d = 0$ or the remaining parallel variation is small.

### *4.1.1 The perpendicular variation*

If the simplex $S$ has positive dimension and codimension, so $0 < d < D$, then moments (21) are simplified by extracting the part of the target vector $t$ perpendicular to the $d$-dimensional affine hyperplane

$$H = \{s = v_0 + \sum_{i=1}^{d} \theta_i(v_i - v_0) \,|\, \theta_i \in R\}$$

containing the domain of integration

$$S = \{v_0 + \sum_{i=1}^{d} \theta_i (v_i - v_0) \,|\, \theta_i \geq 0,\, \sum_{i=1}^{d} \theta_i \leq 1\}. \tag{22}$$

Define the $D \times d$ full-rank matrix $V$ to have columns $v_i - v_0 \in \mathbf{R}^D$ for $i = 1$ to $d$, so that the simplex $S$ is parametrized by $s = v_0 + V\theta$ where $\theta$ varies over the standard $d$-dimensional simplex

$$S_0 = \{(\theta_1, \ldots, \theta_d) \,|\, \theta_i \geq 0,\, \sum_{i=1}^{d} \theta_i \leq 1\}.$$

Then

$$H = \{s = v_0 + V\theta \,|\, \theta \in R^d\}$$

and the moments (21) are given by

$$\begin{aligned} F_\alpha(d, S, f) &= \mathrm{vol}(S) \int_{S_0} \exp(\mathrm{i}t^T(v_0 + V\theta))(v_0 + V\theta - \sigma)^\alpha f(v_0 + V\theta)\, \mathrm{d}\theta \\ &= \mathrm{vol}(S) \exp(\mathrm{i}t_\perp^T v_0) \int_{S_0} \exp(\mathrm{i}t_\parallel^T (v_0 + V\theta)(v_0 + V\theta - \sigma)^\alpha f(v_0 + V\theta)\, \mathrm{d}\theta \end{aligned}$$

where $\mathrm{vol}(S) = \sqrt{\det(V^T V)}$ [27]. Here $t$ has been decomposed into its projections $t_\perp$ and $t_\parallel$, respectively perpendicular to and parallel to the hyperplane $H$ containing $S$. Computationally, $t_\parallel = V(V^T V)^{-1} V^T t$ satisfies an overdetermined least squares problem solved by applying the pseudoinverse or left inverse of $V$. The perpendicular component $t_\perp = t - t_\parallel$ lies in the nullspace of $V^T$ and hence plays no role in the variation of the integrand over the simplex (22). Returning to the original variables thus gives

$$F_\alpha(d, S, f) = \exp(\mathrm{i}t_\perp^T v_0) G_\alpha(d, S, f) \tag{23}$$

where the parallel moment vector $G$ is defined by

$$G_\alpha(d, S, f) = \int_S \exp(\mathrm{i}t_\parallel^T s)(s - \sigma)^\alpha f(s)\, \mathrm{d}s. \tag{24}$$

Henceforth we focus on evaluation of $G$.

### *4.1.2 Small parallel variation*

When $\|V^T t_\parallel\| = \|V^T t\|$ is small, the variation of the exponential factor $\exp(\mathrm{i} t_\parallel^T s)$ in the integrand of $G$ is small, so numerical quadrature will be extremely accurate. For example, if $\|V^T t\| \le \epsilon$ then the exponential factor of the integrand in Eq. (24) can be replaced by $\exp(\mathrm{i} t_\parallel^T v_0)$ with relative error $O(\epsilon)$, by $\exp(\mathrm{i} t_\parallel^T v_0)(1 + \mathrm{i} t_\parallel^T s)$ with relative error $O(\epsilon^2)$, and so forth. It is not necessary explicitly to carry out this replacement, however, because small parallel variation guarantees the accuracy of an appropriate quadrature rule. Such rules have been extensively developed and occur in many variants [28]. Our implementation employs suboptimally accurate but convenient Grundmann-Moeller (GM) rules [29], which can be generated in arbitrary simplex dimension and degree of exactness. A GM rule which is exact for the $Q = \binom{p+m+q+d}{d}$ polynomials of degree $p + m + q$ on $S$ will yield $O(\|V^T t_\parallel\|^{q+1})$ accuracy for $M$ moments up to order $m$, at a cost of $O(QM)$ arithmetic.

Specification of quadrature rules also suggests a specific choice of basis for polynomials $f$ of degree $p$ on a $d$-dimensional simplex in $\mathbf{R}^D$. The Bernstein-Bezier basis, for example, facilitates geometric operations on curved surfaces [30], and has recently been employed in the geometric nonuniform fast Fourier transform [12] and in the finite element method [31]. We employ the convenient and flexible Lagrange representation which parametrizes $f$ by polynomial values at equispaced points in each simplex. Thus a degree-$p$ polynomial $f$ is represented by an $P$-vector of values $f(s_\alpha)$, where $P = \binom{p+d}{d}$ and typical equidistant point sets $s_\alpha$ are shown in Fig. 5.

Lagrange representation simplifies evaluation at the $Q$ equidistant quadrature points of GM rules, and trivializes transformation to the reference simplex $S_0$. In this representation, linear operations such as differentiation, restriction to simplex boundaries, or multiplication by shifted monomials $(s - \sigma)^\alpha$ are straightforward and stable. It is also convenient when the densities $f_j$ are continuous functions such as the exponentials, sinusoids, and logarithms commonly encountered in applications.

### *4.1.3 Dimensional recurrence*

When the variation is not small, we evaluate the parallel moments $G$ in Eq. (23) by a dimensional recurrence based on the Gauss formula for multidimensional integration by parts. Given a vector $h$ parallel to the $d$-dimensional affine hyperplane $H$ containing $S$, and a smooth function $\varphi$ on $\mathbf{R}^D$, the Gauss formula reads

$$\int\limits_S h^T \nabla \varphi(s)\, \mathrm{d}s = \int\limits_{\partial S} h^T n\, \varphi(s)\, \mathrm{d}s,$$

where the boundary $\partial S$ and outward unit normal vector $n$ of $S$ are defined relative to $H$. (See Fig. 6.)

Applying the Gauss formula to the triple product $e(s)f(s)g(s) = \exp(\mathrm{i}t_{\|}^{T}s)f(s)(s-\sigma)^{\alpha}$, where $h^{T}\nabla e(s) = \mathrm{i}h^{T}t_{\|}e(s)$ and $g(s) = (s-\sigma)^{\alpha}$ is a polynomial of degree $|\alpha| \le m$, gives

$$\int_{S} h^{T}\nabla(efg) = \int_{\partial S} h^{T}nefg$$

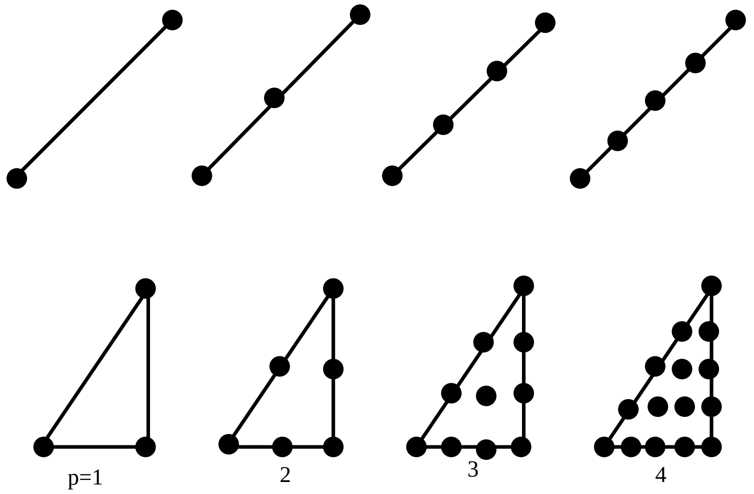


Fig. 5. $P = \binom{p+d}{d} = 2, 3, 4, 5, 3, 6, 10, 15$ equally spaced points for the representation of polynomials of degree $p = 1$ through 4 on simplices of dimension $d = 1$ and 2.

$$= \int_S \mathrm{i}h^T t_\| efg + eh^T \nabla fg + efh^T \nabla g. \tag{25}$$

Solving for $\int_S efg$ gives

$$\int_S efg = \frac{1}{\mathrm{i}h^T t_\|} \left[ \int_{\partial S} h^T nefg - \int_S eh^T \nabla fg - \int_s efh^T \nabla g \right]$$

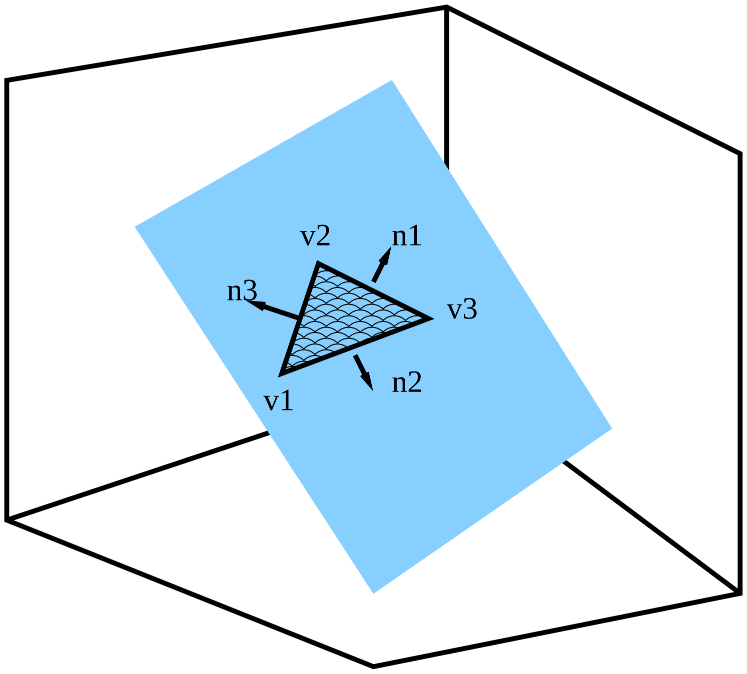


Fig. 6. Vertices $v_k$ and outward unit normal vectors $n_k$ relative to the affine hyperplane $H$, for a simplex with $d = 2$ in ambient dimension $D = 3$.

or

$$\int_S e((I + z^T\nabla)f)g = \int_{\partial S} z^T nefg - \int_S efz^T\nabla g \tag{26}$$

where $z = h/(\mathrm{i}h^T t_{\|})$ and $I$ is the identity operator. Since (26) holds for all polynomials $f$ and $g$, it holds with $f$ replaced by the degree-$p$ polynomial

$$f_1 = \left(I + z^T\nabla\right)^{-1} f = \left(I - z^T\nabla + (z^T\nabla)^2 - \cdots + (-1)^p(z^T\nabla)^p\right) f.$$

The sum terminates because $\deg(f) \leq p$. Here $I + z^T\nabla$ is an invertible linear operator on the $P$-dimensional space of polynomials of degree $\leq p$, since all eigenvalues of the operator $z^T\nabla$ are 0. Since $(I + z^T\nabla)f_1 = f$, Eq. (26) yields

$$\int_S efg = \int_{\partial S} z^T n\ ef_1 g - \int_S ef_1 z^T\nabla g = \int_{\partial S} z^T n\ ef_1 g_0 + \int_S ef_1 g_1 \tag{27}$$

where $g_0 = g$ and $g_k = (-z^T\nabla)^k g$ is a polynomial of degree $\leq m-k$ for $k \geq 1$. Iterating the recurrence relation (27) eventually eliminates the integrals over $S$ completely since $g_{m+1} = 0$, yielding a dimensional recurrence

$$\int_S efg = \int_{\partial S} z^T n\ e\left(f_1 g_0 + f_2 g_1 + \cdots + f_{m+1} g_m\right) \tag{28}$$

where $f_k = (I + z^T\nabla)^{-k} f$ for $k \geq 1$. Each $g_j$ term is a linear combination of moments of $f_{j+1}$ with order $m-j$, so the dimensional recurrence (28) expresses order-$m$ moments of a degree-$p$ polynomial over a simplex $S$ of dimension $d$, as linear combinations of order-$(m-r)$ moments of $(m+1)$ different degree-$p$ polynomials $f_r$ over $d+1$ simplices $\partial_k S$ of dimension $d-1$. Here the boundary $\partial S = \cup_{k=0}^{d} \partial_k S$ consists of $d+1$ oriented simplices $\partial_k S$ of dimension $d-1$, formed by omitting vertex $k = 0$ through $d$ successively.

Define the $D$-vector of moment shift operators $E = (E_1, \ldots, E_D)$ such that

$$z^T\nabla(s-\sigma)^\alpha = z^T E(s-\sigma)^\alpha = \sum_{j=1}^{D} z_j \alpha_j (s-\sigma)^{\alpha - e_j}$$

and

$$z^T E G_\alpha(d, S, f) = \sum_{j=1}^{D} z_j G_{\alpha - e_j}(d, S, f)$$

for $\alpha_j > 0$. Let $G(d, S, f)$ denote the $M$-vector of moments $G_\alpha(d, S, f)$. Then the dimensional recurrence (28) reads explicitly

$$\begin{aligned} G(d,S,f) = & \sum_{k=0}^{d} z^T n_k (G(d-1, \partial_k S, f_1) - z^T E G(d-1, \partial_k S, f_2) \quad (29) \\ & + (-z^T E)^2 G(d-1, \partial_k S, f_3) + \cdots + (-z^T E)^m G(d-1, \partial_k S, f_{m+1})) \\ = & \sum_{k=0}^{d} z^T n_k \sum_{r=0}^{m} (-z^T E)^r G(d-1, \partial_k S, f_{r+1}) \quad (30) \end{aligned}$$

For a single moment of order $|\alpha| = 0$, as in direct evaluation of the Fourier transform (1), the double sum (29) simplifies to

$$G(d, S, f) = \sum_{k=0}^{d} z^T n_k \; G(d-1, \partial_k S, f_1). \quad (31)$$

Thus the direct transform is exactly the sum of $d+1$ direct transforms of degree-$p$ polynomials $f_1$ over $(d-1)$-dimensional simplices $\partial_k S$. The recurrence terminates when the dimension of the simplex reaches 0, as the simplices then become points. Thus the cost of the direct transform on $d$-dimensional simplices is equivalent to $(d+1)!$ *pointwise* direct transforms. (Note that this recurrence does not convert standard fast algorithms for points where $d = 0$ to algorithms for simplices with $d > 0$, because the transformation $f \to f_1$ depends on the target $t$.)

An alternate dimensional recurrence can be obtained by interchanging the roles of $f$ and $g$. Such an interchange leads to a shorter recurrence

$$\int_S efg = \int_{\partial S} z^T n \; e \left( g^1 f^0 + g^2 f^1 + \cdots + g^{p+1} f^p \right) \quad (32)$$

if $p < m$, where now

$$g^k = (I + z^T \nabla)^{-k} g, \qquad f^k = (-z^T \nabla)^k f.$$

Each $g^k$ is a linear combination of order-$m$ moments, so the dimensional recurrence (32) expresses order-$m$ moments of a degree-$p$ polynomial over a simplex $S$ of dimension $d$, as linear combinations of order-$m$ moments of $(p+1)$ degree-$(p-r)$ polynomials $f_r$ over $d+1$ simplices $\partial_k S$ of dimension $d-1$. Explicitly,

$$G(d, S, f) = \sum_{k=0}^{d} z^T n_k ((I + z^T E)^{-1} G(d-1, \partial_k S, f^0) \quad (33)$$

$$
\begin{aligned}
&+ (I + z^T E)^{-2} G(d-1, \partial_k S, f^1) + \cdots + (I + z^T E)^{-p-1} G(d-1, \partial_k S, f^p)) \\
&= \sum_{k=0}^{d} z^T n_k \sum_{r=1}^{p+1} (I + z^T E)^{-r-1} G(d-1, \partial_k S, f^r). \qquad (34)
\end{aligned}
$$

Fig. 7 displays a $G$ evaluation scheme which combines quadrature for small parallel variation with recurrence (34) for large parallel variation.

Fig. 7. Recursive evaluation of $G = G(d, S, f)$ by quadrature and recurrence.

```
if $\|V^T t_\|\| \leq \epsilon$
    Generate quadrature rule of order $p + m$
    Evaluate $G(d, S, f)$ by quadrature
else
    $z = -\mathrm{i} t_\| / \|t_\|\|^2$
    $G = 0$
    $f^0 = f$
    for $r = 1 \ldots p + 1$
        $f^r = (-z^T \nabla) f^{r-1}$
        for $k = 0 \ldots d$
            $G = G + (I + z^T E)^{-r-1} G(d-1, \partial_k S, f^r)$
        end for
    end for
end if
```

### *4.1.4 Computational cost and stability*

The total computational effort $W(m, p, d, D)$ required to evaluate all order-$m$ moments of a degree-$p$ polynomial on a dimension-$d$ simplex in $\mathbf{R}^D$ can now be bounded. Using the dimensional recurrence (28), for example, requires a $P \times P$ matrix-vector multiply to generate each of the $m$ auxiliary polynomials $f^{r+1}$. Then $M$ moments $G(d, S, f)$ for $|\alpha| \leq m$ are linear combinations of $O(M)$ boundary moments of order $m - r$ for $f_{r+1}$ as $r$ ranges from 0 to $m$. Hence the total cost $W(m, p, d, D)$ for evaluating all $M$ moments in simplex dimension $d$ satisfies

$$
W(m, p, d, D) \leq mp^d + O(m^{2D+1}) W(d-1, m) \leq O(mp^d + m^{1+d(2D+1)})
$$

Although the exponent is large, the cost of moment evaluation is much smaller in practice because many branches of the dimensioonal recurrence terminate with quadrature.

Both recurrences (28) and (32) are numerically stable when $z$ is small and the terms in the formally infinite sum defining $(I + z^T \nabla)^{-1}$ rapidly decrease to zero. The natural choice $h = t_\|$ makes $z = -\mathrm{i} t_\| / \|t_\|\|^2$ small when $t_\|$ is large. When the variation $\|t_\|\|$ is small, of course, the recurrence can be highly unstable and numerical integration is employed as in 4.1.2.

## *4.2 Tree structures*

Two additional ingredients, approximation and remaindering, are necessary to localize geometric objects such as simplices into spatial tree structures similar to those of Section 2.2, with cells satisfying inequality (18).

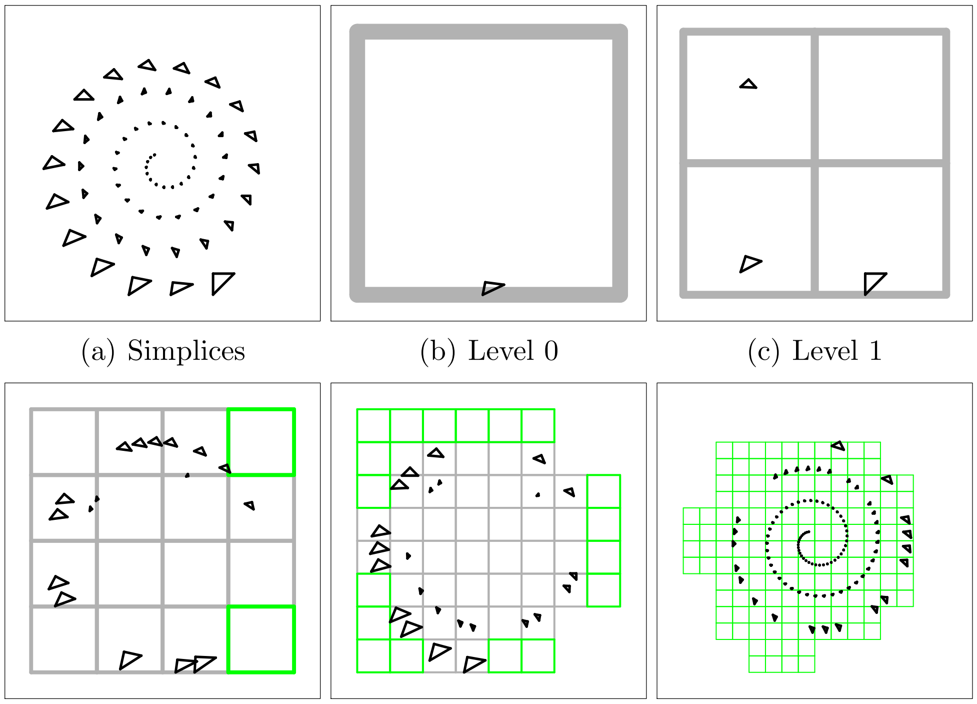
Unlike points, a given simplex may not lie exactly within a given cubical cell $S_{l,j}$. Thus we define a simplex $S$ to lie $\epsilon$-approximately within a cell $\sigma + RQ$ if each vertex $v$ of $S$ satisfies $|v_j - \sigma_j| \leq (1+\epsilon)R$ for $j = 1$ to $D$. Our tree construction procedure initially assigns all simplices to the root cell $S_{0,0}$. Then simplices are $\epsilon$-approximately assigned to child cells when possible.

However, some simplices may be too large or too awkwardly placed to fit into any cell on a given level $l$. See Figs. 8 and 9 for examples. Such simplices simply remain in the smallest possible source cell, and contribute into the coefficients later. (They can additionally be subdivided by the algorithm of [12]). The butterfly algorithm collects these simplices at each level as it works its way up and down the source and target trees. After each split-merge step, source simplices remaining in each source cell $S_{l-1,I}$ are folded into the coefficient vector for each target cell $T_{L-l+1,j}$ via

$$\begin{aligned} C_\alpha(\sigma_{l-1,I}, \tau_{L-l+1,j}) = {} & C_\alpha(\sigma_{l-1,I}, \tau_{L-l+1,j}) \\ & + \sum_{S_k \subset S_{l-1,I}} \exp(\mathrm{i}\tau_{L-l+1,j}^T \sigma_{l-1,I}) \frac{\mathrm{i}^{|\alpha|}}{\alpha!} F_\alpha(d, S_k, f_j). \end{aligned} \tag{35}$$

The cost of the algorithm is unaffected by this modification if the number of remaindered simplices is $O(1)$.

A detailed piecewise-polynomial butterfly algorithm is exhibited in Fig. 10.


(a) Simplices (b) Level 0 (c) Level 1

(d) Level 2 (e) Level 3 (f) Level 4

Fig. 8. A spatial tree structure for geometric data with $d = D = 2$ and $L = 4$ levels. Simplices of various sizes are sorted into cells of a tree structure with an overlap tolerance of 5 percent, leaving a few awkwardly placed simplices in non-leaf cells on every level. Almost all of the simplices are $\epsilon$-approximately assigned to leaf cells on Level 4.

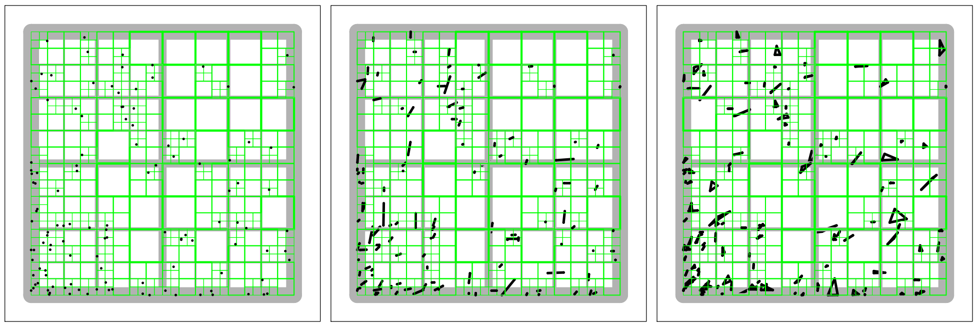

(a) Points (b) Segments (c) Triangles

Fig. 9. Spatial tree structures for points, segments and triangles

Fig. 10. A piecewise-polynomial butterfly algorithm

**Step 1 - Localization**

Sort source simplices $S_j$ and target points $t_k$ into *minimal* cells of $L$-level trees $\mathcal{S}_L$ and $\mathcal{T}_L$

**Step 2 - Compute source-centered coefficients**

```
for $j = 0 \dots 2^{DL} - 1$
    $S = S_{L,j}$
    $\sigma = \sigma_{L,j}$
    $\tau = \tau_{0,0}$
    for $|\alpha| \le m$
        $C_\alpha(\sigma, \tau) = \exp(\mathrm{i}\tau^T\sigma)\frac{\mathrm{i}^{|\alpha|}}{\alpha!}\sum_{S_k \subset S}\int_{S_k}(s-\sigma)^\alpha \exp(\mathrm{i}\tau^T(s-\sigma))f_k(s)\,\mathrm{d}s$
    end for
end for
```

**Step 3 - Butterfly with remainder**

```
for $l = L \dots 1$
    for $I = 0 \dots 2^{Dl} - 1$
        $S = S_{l-1,I}$
        $\sigma = \sigma_{l-1,I}$
        for $J = 0 \dots 2^{D(L-l)} - 1$
            $\tau = \tau_{L-l,J}$
            for $j = 0 \dots 2^D - 1$
                $\tau_1 = \tau_{L-l+1,2^D J+j}$
                $C(\sigma, \tau_1) = \sum_{i=0}^{2^D-1} L(\sigma_{l,I/2^D+i} - \sigma, \tau_1)R(\sigma_{l,I/2^D+i}, \tau - \tau_1)C(\sigma_{l,I/2^D+i}, \tau)$
                for $S_k \subset S$
                    for $|\alpha| \le m$
                        $C_\alpha(\sigma, \tau_1) = C_\alpha(\sigma, \tau_1) + \exp(\mathrm{i}\tau_1^T\sigma)\frac{\mathrm{i}^{|\alpha|}}{\alpha!}\int_{S_k}(s-\sigma)^\alpha \exp(\mathrm{i}\tau_1^T(s-\sigma))f_k(s)\,\mathrm{d}s$
                    end for
                end for
            end for
        end for
    end for
end for
```

**Step 4 - Evaluate target-centered expansions**

```
for $j = 0 \dots 2^{DL} - 1$
    $\sigma = \sigma_{0,0}$
    $T = T_{L,j}$
    $\tau = \tau_{L,j}$
    for $t_k \in T_{L,j}$
        $\widehat{f}(t_k) = \exp(\mathrm{i}(t_k - \tau)^T\sigma)) \sum_{|\alpha| \le m} C_\alpha(\sigma, \tau)(t_k - \tau)^\alpha$
    end for
end for
```

## 5 Numerical results

The piecewise-polynomial butterfly algorithm (10) has been implemented in the Fortran 77 programming language, for simplices of arbitrary dimension $d$ in arbitrary ambient dimension $D$, with multidimensional arrays mapped to one-dimensional arrays. Linear operations such as differentiation, integration and interpolation of polynomials are implemented by matrix-vector multiplications with precomputed matrices.

As a result of this dimensional generality, our present implementation is sub-optimally efficient, and all timing results should be viewed as preliminary. We plan to investigate speedups such as parallelization in future work.

The accuracy and efficiency of the implementation have been verified on a gallery of test cases including simplices of dimensions 0 through 2 in ambient dimensions 1 through 3. All results below were obtained with the `gfortran` compiler on a single Intel Xeon E5-2670 v2 processor with 64 GB of memory.

### *5.1 Discrete points in* $\mathbf{R}^D$

Fig. (10) evaluates the pointwise nonuniform Fast Fourier Transform (3) in the case when the source simplex dimension $d = 0$. We tested the pointwise transform on $N$ random source and target points distributed over source and target domains of size $O(1)$ and $O(N)$ respectively. Table 1 shows the CPU times $T_s$ required to achieve $s = 3, 6, 9$ and 12-digit accuracy, as well as the CPU time $T_d$ required by direct evaluation. The CPU time $T_F$ required for a standard $N_F$-point complex $D$-dimensional FFT [32] is also shown. The FFT timings are monotonized by choosing $N_F$ to be the smallest product of powers of 2, 3 and 5 which is larger than $N$. Tables 2 and 3 show the CPU times $T_s$ for $D = 2$ and $D = 3$, with asterisks indicating cases where the butterfly algorithm required more memory than the 4 GB installed in our current workstation. We conclude that the pointwise butterfly algorithm is much faster than direct evaluation even for small problem sizes $N$ and maximum twelve-digit accuracy, in $D = 1$ dimensions. In $D = 2$ dimensions it is faster for all problem sizes $N$ at three-digit accuracy, but twelve-digit accuracy does not break even until $N = 8100$. In $D = 3$ dimensions it is faster for all problem sizes $N$ at three-digit accuracy, while six-digit accuracy does not break even until $N = 5832$. The memory limitations of our current workstation slow the butterfly algorithm considerably for higher accuracy in three dimensions, so that it never breaks even with direct evaluation if more than six digits are requested. Hence our higher-accuracy results in two and three dimensions tend to understate the performance of the butterfly algorithm; it should run orders of magnitude faster in less constrained computing environments.

Compared to the classical FFT for $N_F$ uniform points, the butterfly algorithm is asymptotically within two orders of magnitude for three-digit accuracy. Since our implementation operates in arbitrary simplex dimension and ambient dimension, an optimized implementation might run ten times faster, and compare more favorably with a standard uniform FFT. The cost of the butterfly algorithm is a polylogarithmic function of the accuracy, and our results suggest that doubling the accuracy less than doubles the cost (in $D = 1$).

Table 1
CPU times $T_s$ for $s = 3$, 6, 9 and 12-digit accuracy, for $N$ random points with $d = 0$ in ambient dimension $D = 1$.

| $N$ | $T_F$ | $T_d$ | $T_3$ | $T_6$ | $T_9$ | $T_{12}$ |
|---|---|---|---|---|---|---|
| 729 | 0.00008 | 0.087 | 0.006 | 0.009 | 0.012 | 0.020 |
| 5832 | 0.00063 | 5.714 | 0.057 | 0.098 | 0.132 | 0.216 |
| 10935 | 0.00120 | 19.789 | 0.110 | 0.183 | 0.256 | 0.329 |
| 16038 | 0.00180 | 42.655 | 0.162 | 0.254 | 0.413 | 0.580 |
| 21141 | 0.00240 | 74.195 | 0.231 | 0.392 | 0.513 | 0.703 |
| 26244 | 0.00300 | 113.856 | 0.278 | 0.451 | 0.629 | 1.026 |
| 31347 | 0.00359 | 163.317 | 0.359 | 0.554 | 0.931 | 1.163 |
| 36450 | 0.00439 | 224.520 | 0.430 | 0.678 | 0.998 | 1.391 |
| 41553 | 0.00498 | 290.851 | 0.502 | 0.772 | 1.107 | 1.537 |
| 46656 | 0.00537 | 369.967 | 0.528 | 0.875 | 1.225 | 2.014 |

Table 2
CPU times $T_s$ for $s = 3$, 6, 9 and 12-digit accuracy, for $N$ random points with $d = 0$ in ambient dimension $D = 2$.

| $N$ | $T_F$ | $T_d$ | $T_3$ | $T_6$ | $T_9$ | $T_{12}$ |
|---|---|---|---|---|---|---|
| 729 | 0.00018 | 0.100 | 0.035 | 0.158 | 0.418 | 0.938 |
| 2304 | 0.00048 | 0.990 | 0.087 | 0.481 | 1.665 | 3.464 |
| 4761 | 0.00098 | 4.091 | 0.165 | 1.056 | 2.974 | 7.028 |
| 8100 | 0.00178 | 11.865 | 0.281 | 1.522 | 4.813 | 11.322 |
| 12321 | 0.00273 | 28.155 | 0.394 | 2.723 | 7.261 | 15.510 |
| 17424 | 0.00391 | 55.471 | 0.681 | 3.100 | 10.457 | 23.183 |
| 23409 | 0.00547 | 100.128 | 0.717 | 4.496 | 13.668 | 33.479 |
| 30276 | 0.00703 | 166.163 | 0.869 | 5.145 | 18.234 | 39.695 |
| 38025 | 0.00937 | 259.937 | 0.945 | 6.598 | 20.500 | 53.227 |
| 46656 | 0.01250 | 408.240 | 1.578 | 8.313 | 29.188 | 61.246 |

Table 3
CPU times $T_s$ for $s = 3$, 6, 9 and 12-digit accuracy, for $N$ random points with $d = 0$ in ambient dimension $D = 3$.

| $N$ | $T_F$ | $T_d$ | $T_3$ | $T_6$ | $T_9$ | $T_{12}$ |
|---|---|---|---|---|---|---|
| 729 | 0.00035 | 0.114 | 0.129 | 0.816 | 5.859 | 23.207 |
| 1728 | 0.00085 | 0.675 | 0.242 | 3.063 | 12.023 | 50.125 |
| 3375 | 0.00347 | 2.637 | 0.344 | 4.563 | 21.141 | 77.188 |
| 5832 | 0.00586 | 6.379 | 1.031 | 6.516 | 29.531 | 191.641 |
| 9261 | 0.01875 | 17.364 | 1.250 | 8.563 | 73.594 | 248.813 |
| 13824 | 0.01875 | 38.880 | 1.938 | 20.438 | 92.281 | 4219.156 |
| 19683 | 0.03750 | 73.811 | 2.375 | 26.844 | 290.906 | *** |
| 27000 | 0.03750 | 151.875 | 2.813 | 31.375 | 565.625 | *** |
| 35937 | 0.03750 | 247.067 | 3.125 | 37.500 | 615.000 | *** |
| 46656 | 0.03750 | 408.240 | 3.438 | 41.313 | *** | *** |

## *5.2 Line segments in* $\mathbf{R}^D$

We tested the butterfly algorithm on $n$ randomly placed line segments in $\mathbf{R}^D$, setting each $f_j$ to a random cubic polynomial and evaluating $\widehat{f}(t_k)$ at $N$ random points $t_k \in \mathbf{R}^D$. Here $N$ is the total number of degrees of freedom in the input, which for cubic polynomials on line segments is $4n$, and $N_F$ is again the smallest multiple of powers of 2, 3 and 5 which is larger than $N$.

As in the case of points where $d = 0$, the butterfly algorithm compares very favorably with direct evaluation. For six-digit accuracy it beats direct evaluation for every problem size $N$, by three orders of magnitude when $N = 186624$. Since the number of degrees of freedom is larger, the standard FFT has more work to do and the butterfly algorithm does better by comparison. For three-digit accuracy in $D \geq 1$, it runs as fast as 25 FFTs.

Table 4
CPU times $T_s$ for $s$ = 3, 6, 9 and 12-digit accuracy, for cubic polynomials on $n = N/4$ random segments with $d = 1$ in ambient dimension $D = 1$.

| $N$ | $T_F$ | $T_d$ | $T_3$ | $T_6$ | $T_9$ | $T_{12}$ |
|---|---|---|---|---|---|---|
| 2916 | 0.00030 | 0.948 | 0.006 | 0.010 | 0.014 | 0.022 |
| 23328 | 0.00260 | 60.819 | 0.060 | 0.098 | 0.136 | 0.228 |
| 43740 | 0.00500 | 214.415 | 0.125 | 0.205 | 0.286 | 0.356 |
| 64152 | 0.00740 | 460.701 | 0.189 | 0.294 | 0.446 | 0.574 |
| 84564 | 0.00981 | 798.362 | 0.235 | 0.397 | 0.581 | 0.757 |
| 104976 | 0.01240 | 1230.700 | 0.321 | 0.507 | 0.692 | 1.080 |
| 125388 | 0.01475 | 1760.514 | 0.390 | 0.622 | 0.842 | 1.218 |
| 145800 | 0.01660 | 2375.657 | 0.430 | 0.743 | 1.065 | 1.355 |
| 166212 | 0.01992 | 3084.012 | 0.503 | 0.847 | 1.192 | 1.499 |
| 186624 | 0.02109 | 3891.037 | 0.527 | 0.956 | 1.323 | 2.074 |

Table 5
CPU times $T_s$ for $s = 3$, 6, 9 and 12-digit accuracy, for cubic polynomials on $N$ random segments with $d = 1$ in ambient dimension $D = 2$.

| $N$ | $T_F$ | $T_d$ | $T_3$ | $T_6$ | $T_9$ | $T_{12}$ |
|---|---|---|---|---|---|---|
| 2916 | 0.00059 | 1.068 | 0.024 | 0.118 | 0.409 | 0.865 |
| 9216 | 0.00195 | 10.598 | 0.090 | 0.464 | 1.201 | 2.739 |
| 19044 | 0.00391 | 45.378 | 0.176 | 0.742 | 2.500 | 6.881 |
| 32400 | 0.00703 | 134.156 | 0.223 | 1.299 | 4.814 | 10.311 |
| 49284 | 0.01172 | 302.731 | 0.412 | 2.041 | 7.145 | 14.998 |
| 69696 | 0.01641 | 607.117 | 0.455 | 2.979 | 9.045 | 22.514 |
| 93636 | 0.02031 | 1093.639 | 0.770 | 4.555 | 13.367 | 28.801 |
| 121104 | 0.02813 | 1842.578 | 0.832 | 5.066 | 14.195 | 38.488 |
| 152100 | 0.03750 | 2954.364 | 0.973 | 5.391 | 20.215 | 41.859 |
| 186624 | 0.04219 | 4441.432 | 1.082 | 6.969 | 24.133 | 60.227 |

Table 6
CPU times $T_s$ for $s = 3$, 6, 9 and 12-digit accuracy, for cubic polynomials on $N$ random segments with $d = 1$ in ambient dimension $D = 3$.

| $N$ | $T_F$ | $T_d$ | $T_3$ | $T_6$ | $T_9$ | $T_{12}$ |
|---|---|---|---|---|---|---|
| 2916 | 0.00142 | 1.196 | 0.133 | 0.766 | 3.469 | 12.797 |
| 6912 | 0.00361 | 6.885 | 0.188 | 1.336 | 11.797 | 25.906 |
| 13500 | 0.00469 | 26.631 | 0.359 | 3.523 | 20.883 | 74.656 |
| 23328 | 0.00937 | 80.190 | 0.484 | 6.359 | 29.500 | 107.219 |
| 37044 | 0.01875 | 199.690 | 0.578 | 8.641 | 74.047 | 229.078 |
| 55296 | 0.01875 | 444.960 | 1.563 | 10.344 | 48.969 | 687.000 |
| 78732 | 0.03750 | 898.037 | 1.813 | 12.594 | 112.000 | 4574.969 |
| 108000 | 0.03750 | 1704.375 | 2.813 | 15.188 | 329.375 | *** |
| 143748 | 0.03750 | 3009.724 | 3.250 | 39.563 | 609.938 | *** |
| 186624 | 0.03750 | 5073.840 | 3.813 | 45.625 | 4399.375 | *** |

## *5.3 Triangles in* $\mathbf{R}^D$

We also tested the butterfly algorithm on $n$ randomly placed triangles in $\mathbf{R}^D$, setting each $f_j$ to a random cubic polynomial and evaluating $\widehat{f}(t_k)$ at $N$ random points in $\mathbf{R}^D$. Here $N$ is the total number of degrees of freedom in the input, which for cubic polynomials on triangles is $10n$.

As in the case of segments, the butterfly algorithm compares very favorably with direct evaluation. For six-digit accuracy it beats direct evaluation for every problem size $N$, by three orders of magnitude when $N = 466560$. Furthermore, the butterfly algorithm requires only 10 FFTs to obtain three-digit accuracy in ambient dimension $D = 2$.

Table 7
CPU times $T_s$ for $s = 3$, 6, 9 and 12-digit accuracy, for cubic polynomials on $n = N/10$ random triangles with $d = 2$ in ambient dimension $D = 2$.

| $N$ | $T_F$ | $T_d$ | $T_3$ | $T_6$ | $T_9$ | $T_{12}$ |
|---|---|---|---|---|---|---|
| 7290 | 0.00154 | 6.400 | 0.029 | 0.133 | 0.438 | 0.943 |
| 23040 | 0.00562 | 66.414 | 0.070 | 0.434 | 1.285 | 2.869 |
| 47610 | 0.01045 | 274.862 | 0.145 | 0.807 | 2.405 | 5.791 |
| 81000 | 0.01992 | 786.507 | 0.264 | 1.410 | 4.948 | 11.447 |
| 123210 | 0.03047 | 1837.802 | 0.323 | 2.271 | 6.012 | 15.571 |
| 174240 | 0.04219 | 3651.893 | 0.391 | 3.201 | 9.545 | 22.889 |
| 234090 | 0.06094 | 6772.151 | 0.676 | 3.623 | 13.414 | 32.809 |
| 302760 | 0.07969 | 11314.472 | 1.098 | 5.680 | 15.449 | 36.672 |
| 380250 | 0.10938 | 18017.314 | 1.215 | 6.094 | 22.102 | 50.063 |
| 466560 | 0.13125 | 27191.700 | 1.344 | 7.688 | 23.680 | 64.141 |

Table 8
CPU times $T_s$ for $s = 3$, 6, 9 and 12-digit accuracy, for cubic polynomials on $n = N/10$ random triangles with $d = 2$ in ambient dimension $D = 3$.

| $N$ | $T_F$ | $T_d$ | $T_3$ | $T_6$ | $T_9$ | $T_{12}$ |
|---|---|---|---|---|---|---|
| 7290 | 0.00361 | 7.233 | 0.063 | 0.797 | 3.797 | 13.508 |
| 17280 | 0.00000 | 41.040 | 0.219 | 1.406 | 6.859 | 27.203 |
| 33750 | 0.01875 | 155.039 | 0.297 | 4.672 | 18.469 | 79.641 |
| 58320 | 0.01875 | 472.027 | 0.500 | 6.500 | 29.938 | 111.938 |
| 92610 | 0.01875 | 1172.095 | 0.656 | 9.000 | 39.406 | 253.781 |
| 138240 | 0.03750 | 2609.280 | 0.844 | 11.219 | 51.781 | 3070.313 |
| 196830 | 0.07500 | 5265.203 | 2.063 | 14.188 | 116.500 | 4854.438 |
| 270000 | 0.11250 | 9973.125 | 2.438 | 17.188 | 342.938 | *** |
| 359370 | 0.15000 | 17182.378 | 2.688 | 21.438 | 364.063 | *** |
| 466560 | 0.18750 | 28343.520 | 4.000 | 24.063 | 3126.563 | *** |

## 6 Extensions and applications

The algorithm described above can be accelerated, generalized and applied in several ways. We discuss acceleration by change of basis via low-rank approximation, extension to the evaluation of Galerkin matrix elements, and generalization to Laplace and Gauss transforms.

### *6.1 Acceleration*

A standard ingredient of many other butterfly algorithms is a choice of basis functions which accelerates the algorithm. For example, Taylor series approximation of the exponential is suboptimal for approximation on a real interval. Minimax or Chebyshev approximation can attain equal accuracy with fewer terms, leading to a factor of 2 speedup in the one-dimensional pointwise butterfly algorithm of [15].

Often the basis functions are chosen implicitly by transforming the shift and merge matrices in the intermediate steps. Butterfly algorithms can be thought of as matrix factorizations [22], where the basic matrix $A$ with elements

$$A_{jk} = \exp \mathrm{i} t_j^T s_k$$

is written as

$$A = E B_1 B_2 \cdots B_L F.$$

Here $F$ forms source values into coefficients, $E$ evaluates approximate values at targets, and the intermediate matrices $B_j$ are sparse block matrices that perform shift and merge operations on the coefficients. Most of the computational effort goes into applying the intermediate factors $B_j$, so economizing $B_j$ pays large rewards. For example, consider diagonalizing each block of $B_j$ as in [33,34]. Then shift and merge operations become diagonal, reducing their cost substantially from $O(M^2)$ to $O(M)$. However, the transformation to diagonal form is difficult to implement in an efficient stable form. Thus popular alternatives tend to employ factorizations such as the SVD or interpolative decompositions.

Interpolative decompositions factorize

$$B_j = XYZ$$

with $X$ and $Z$ submatrices of $B_j$ and $Y$ small and well-conditioned. They speed up the application of $B_j$ to arbitrary vectors, and offer considerable speedups in the butterfly algorithm. We plan to incorporate such factorizations into future versions of our code.

### *6.2 Galerkin matrix elements*

In many applications, pointwise evaluation of the Fourier transform is less desirable than the application of Galerkin matrix elements

$$\widehat{f}_{kj} = \int_{T_k} g_k(t) \int_{S_j} \exp(\mathrm{i}t^T s) f_j(s)\,\mathrm{d}s\,\mathrm{d}t, \tag{36}$$

where $T_k \subset \mathbf{R}^D$ are $d$-dimensional target simplices and $g_k$ are degree-$p$ polynomials. Our algorithm extends immediately to the fast application of these Galerkin matrices. The Taylor expansion (11) separates the variables $t$ and $s$ so that

$$\begin{aligned}\widehat{f}_{kj} = {} & \exp(\mathrm{i}\tau^T\sigma) \sum_{|\alpha|\le m} \frac{\mathrm{i}^{|\alpha|}}{\alpha!} \\ & \int_{T_k} (t-\tau)^\alpha \exp(\mathrm{i}(t-\tau)^T\sigma) g_k(t)\,\mathrm{d}t \\ & \int_{S_j} (s-\sigma)^\alpha \exp(\mathrm{i}\tau^T(s-\sigma)) f_j(s)\,\mathrm{d}s + F_m\end{aligned} \tag{37}$$

with controllable error $F_m$. Summing over $j$ yields the same source-centered coefficient vectors as Fig. (10). After the butterfly of Section 4 produces target-centered coefficient vectors, the final evaluation step of Fig. (10) is replaced by a dimensional recurrence identical to (28) with sources replaced by targets.

The resulting algorithm requires a slightly strengthened version of inequality (10). Since there are simplices rather than points in both target space and source space, the rank-$M$ kernel approximation (11) maintains accuracy only if

$$|(t-\tau)^T(s-\sigma)| \le R \quad \text{where} \quad \left(\frac{R\mathrm{e}}{m}\right)^m \le \epsilon \tag{38}$$

whenever $S$ is a source tree cell with center $\sigma$, $T$ is a target tree cell with center $\tau$, $T_k \subset T$, $S_j \subset S$, $t \in T_k$ and $s \in S_j$. Roughly speaking, kernel approximation fails if there are large simplices in both source and target space. This is natural since large simplices encounter many wavelengths of the oscillatory kernel $\exp(\mathrm{i}t^T s)$, and thus require high-frequency approximation. This restriction can be eliminated by simplex subdivision [12].

## *6.3 Laplace and Gauss transforms*

Counterbalancing the potential speedups available from optimized basis functions is a loss of generality. For example, Taylor expansion of the exponential is optimal over disks rather than intervals. Hence our algorithm extends immediately to evaluate the fast Laplace transform [19], where $\mathrm{i} = \sqrt{-1}$ is replaced by $-1$.

The pointwise Gauss transform [17,18]

$$\widehat{f}(t_k) = \sum_{j=1}^{N} \exp(-\|t_k - s_j\|^2) f_j, \qquad 1 \le k \le N, \tag{39}$$

where $\|t\|$ is the Euclidean norm, can be viewed as a pre- and post-processed Laplace transform since

$$\exp(-\|t_k - s_j\|^2) = \exp(-\|t_k\|^2) \exp(2t_k^T s_j) \exp(-\|s_j\|^2).$$

Hence our piecewise-polynomial butterfly algorithm immediately extends to evaluate the piecewise-polynomial Gauss transform with matrix elements

$$\widehat{f}_{kj} = \int\limits_{T_k} g_k(t) \int\limits_{S_j} \exp(-\|t - s\|^2) f_j(s)\, \mathrm{d}s\, \mathrm{d}t. \tag{40}$$

**Acknowledgements**

This material benefited from conversations with Qing-Huo Liu, Eric Michielssen, Vladimir Rokhlin, Ian Sammis, Christopher Wong, and Lexing Ying. This material is based upon work supported by the National Science Foundation under grant number DMS-0913695, and by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number FA9550-11-1-0242, and the SAVIO computational cluster resource provided by the Berkeley Research Computing program at the University of California Berkeley (Supported by UC Chancellor, UC Berkeley Vice Chancellor of Research and Office of the CIO). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

## References

[1] C. F. Gauss, Nachlass: Theoria interpolationis methodo nova tracta, in: Carl Friedrich Gauss, Werke, Band 3, Königlichen Gesellschaft der Wissenschaften, Göttingen, 1866, pp. 265–303.

[2] J. W. Cooley, J. W. Tukey, An algorithm for the machine calculation of complex Fourier series, Math. Comput. 19 (1965) 297–301.

[3] J. Strain, Fast potential theory II: Layer potentials and discrete sums, J. Comput. Phys. 99 (1992) 251–270.

[4] A. Dutt, V. Rokhlin, Fast Fourier transforms for nonequispaced data, SIAM J. Sci. Comput. 14 (6) (1993) 1368–1393.

[5] A. Dutt, V. Rokhlin, Fast Fourier transforms for nonequispaced data. II, Appl. Comput. Harmon. Anal. 2 (1) (1995) 85–100.

[6] G. Beylkin, On the fast Fourier transform of functions with singularities, Appl. Comput. Harm. Analysis 2 (1995) 363–381.

[7] D. Potts, G. Steidl, M. Tasche, Fast Fourier transforms for nonequispaced data: a tutorial, in: Modern sampling theory, Birkhauser, 2001, pp. 247–270.

[8] L. Greengard, J.-Y. Lee, Accelerating the nonuniform fast Fourier transform, SIAM Review 46 (2004) 443–454.

[9] J. Sun, H. Li, Generalized Fourier transform on an arbitrary triangular domain, Adv. Comp. Math. 22 (2005) 223–248.

[10] E. Sorets, Fast Fourier transforms of piecewise constant functions, J. Comput. Phys. 116 (1995) 369–379.

[11] G.-X. Fan, Q. H. Liu, Fast Fourier transform for discontinuous functions, IEEE Trans. Antennas Propag. 52 (2004) 461–465.

[12] I. Sammis, J. Strain, A geometric nonuniform fast Fourier transform, J. Comput. Phys. 228 (2009) 7086–7108.

[13] Y. H. Liu, Q. H. Liu, Z. P. Nie, Z. Q. Zhao, Discontinuous fast Fourier transform with triangle mesh for two-dimensional discontinuous functions, J. Electrom. Waves Appls. 25 (2011) 1045–1057.

[14] E. Michielssen, A. Boag, A multilevel matrix decomposition algorithm for analyzing scattering from large structures, IEEE Trans. Antennas Propagat. 44 (1996) 1086–1093.

[15] M. O'Neil, F. Woolfe, V. Rokhlin, An algorithm for the rapid evaluation of special function transforms, Appl. Comput. Harmon. Anal. 28 (2010) 203–226.

[16] H. Cai, Y. Xu, A fast Fourier-Galerkin method for solving singular boundary integral equations, SIAM J. Num. Analysis 46 (2008) 1965–1984.

[17] L. Greengard, J. Strain, The fast Gauss transform, SIAM J. Sci. Stat. Comput 12 (1991) 79–94.

[18] J. Strain, The fast Gauss transform with variable scales, SIAM J. Sci. Stat. Comput 12 (1991) 1131–1139.

[19] J. Strain, A fast Laplace transform based on Laguerre functions, Math. Comp. 58 (1992) 275–284.

[20] V. Rokhlin, Rapid solution of integral equations of classical potential theory, J. Comput. Phys. 60 (1985) 187–207.

[21] J. Carrier, L. Greengard, V. Rokhlin, A fast adaptive multipole method for particle simulations, SIAM J. Sci. Stat. Comput. 9 (1988) 669–686.

[22] Y. Li, H. Yang, E. R. Martin, K. L. Ho, L. Ying, Butterfly factorization, SIAM J. Multiscale. Mod. Simul. to appear (2015) 1–19.

[23] H. Samet, The design and analysis of spatial data structures, Addison-Wesley, Reading, Massachusetts, 1990.

[24] L. Demanet, M. Ferrara, N. Maxwell, J. Poulson, L. Ying, A butterfly algorithm for synthetic aperture radar imaging, SIAM J. Imag. Sci. 5 (2012) 203–243.

[25] J. Poulson, L. Demanet, N. Maxwell, L. Ying, A parallel butterfly algorithm, SIAM J. Sci. Comput. 36 (2013) C49–C65.

[26] Y. Zhang, J. Liu, E. Kultursay, M. Kandemir, N. Pitsianis, X. Sun, Scalable parallelization strategies to accelerate NuFFT data translation on multicores, in: EuroPar 2010, Part II, LNCS 6272, Vol. 6272 of LNCS, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 125–136.

[27] A. Ben-Israel, The change of variables formula using matrix volume, SIAM J. Matrix Analysis 21 (1999) 300–312.

[28] P. J. Davis, P. Rabinowitz, Methods of Numerical Integration, 2nd Edition, Computer science and applied mathematics, Academic Press, 1984.

[29] A. Grundmann, M. Moeller, Invariant integration formulas for the N-Simplex by combinatorial methods, SIAM J. Num. Analysis 15 (1978) 282–290.

[30] R. T. Farouki, The Bernstein polynomial basis: A centennial retrospective, Comput. Aided Geom. Des. 29 (2012) 379–419.

[31] M. Ainsworth, G. Andriamaro, O. Davydov, Bernstein-Bézier finite elements of arbitrary order and optimal assembly procedures, SIAM J. Sci. Comput. 33 (2011) 3087–3109.

[32] P. N. Swartztrauber, FFTPACK – a package of Fortran subprograms for the fast Fourier transform of periodic and other symmetric sequences, software, National Center for Atmospheric Research (1985).

[33] V. Rokhlin, Diagonal forms of translation operators for the Helmholtz equation in three dimensions, Appl. Comput. Harmon. Anal. 1 (1) (1993) 82–93.

[34] V. Rokhlin, Sparse diagonal forms for translation operators for the Helmholtz equation in two dimensions, Appl. Comput. Harmon. Anal. 5 (1) (1998) 36–67.

# Bilinear quadratures for inner products

Christopher A. Wong

September 29, 2015

**Abstract**

A bilinear quadrature numerically evaluates a continuous bilinear map, such as the $L^2$ inner product, on continuous $f$ and $g$ belonging to known finite-dimensional function spaces. Such maps arise in Galerkin methods for differential and integral equations. The construction of bilinear quadratures over arbitrary domains in $\mathbb{R}^d$ is presented. In one dimension, integration rules of this type include Gaussian quadrature for polynomials and the trapezoidal rule for trigonometric polynomials as special cases. A numerical procedure for constructing bilinear quadratures is developed and validated.

## 1 Introduction

Classical quadratures such as Gaussian and trapezoidal rules accurately evaluate continuous linear functionals such as

$$\int_\Omega f(x)w(x)\,dx$$

for $f$ in a finite-dimensional space of continuous functions. Bilinear quadratures evaluate continuous bilinear forms such as the weighted $L^2$ inner product

$$\langle f,g\rangle_{L^2} = \int_\Omega f(x)g(x)w(x)\,dx$$

or the weighted $H^1$ inner product

$$\langle f,g\rangle_{H^1} = \int_\Omega \sum_{i,j=1}^{d} \Big(\frac{\partial f}{\partial x_i} a_{ij}(x) \frac{\partial g}{\partial x_j}\Big) + f(x)g(x)\,dx$$

on finite-dimensional spaces of continuous functions $f, g$ on $\Omega \subset \mathbb{R}^d$.

$L^2$ inner products compute orthogonal projections onto subspaces, while $H^1$ inner products provide local solutions to elliptic problems, a key ingredient of the finite element method. For example, let $\Omega \subset \mathbb{R}^d$ be a smooth bounded domain, let $L$ be a uniformly elliptic operator, $f \in L^2(\Omega)$, $g \in L^2(\partial\Omega)$, and $\gamma \in L^\infty(\partial\Omega)$. Consider the Robin problem

$$\begin{cases} \text{Find } u \in H^1(\Omega) \text{ satisfying} \\ Lu = f \text{ in } \Omega, \gamma u + \frac{\partial u}{\partial n} = g \text{ on } \partial\Omega. \end{cases} \tag{1.1}$$

When $L = -\Delta$, then if a bilinear form $a : H^1(\Omega) \times H^1(\Omega) \to \mathbb{R}$ is defined by $a(u,v) = \int_\Omega Du \cdot Dv + \int_{\partial\Omega} \gamma uv$, the weak formulation to (1.1) seeks $u \in H^1(\Omega)$ satisfying

$$a(u,v) = \langle f,v\rangle_{L^2(\Omega)} + \langle g,v\rangle_{L^2(\partial\Omega)} \text{ for all } v \in H^1(\Omega). \tag{1.2}$$

The Galerkin method constructs an approximate solution to (1.2) by choosing finite-dimensional function spaces $\mathcal{F}_0, \mathcal{G}_0$ and seeking $u_0 \in \mathcal{F}_0$ satisfying

$$a(u_0,v_0) = \langle f,v_0\rangle_{L^2(\Omega)} + \langle g,v_0\rangle_{L^2(\partial\Omega)} \text{ for all } v_0 \in \mathcal{G}_0. \tag{1.3}$$

The linear system (1.3) is solved in a basis, which requires computing a number of $L^2$ inner product integrals. These integrals should be computed both efficiently and accurately.

Efficiency is achieved by using the fewest function evaluations possible. When $d > 1$, the optimal efficiency of a classical quadrature is unknown. For a bilinear quadrature, the minimum number of function evaluations is equal to the dimension of function space being integrated. The inner product of two functions $f, g$ belonging to given finite-dimensional function spaces is computed by the formula

$$\langle f, g\rangle = f(\mathbf{x})^* W g(\mathbf{y}), \tag{1.4}$$

where $f(\mathbf{x}) \in \mathbb{R}^m$ and $g(\mathbf{y}) \in \mathbb{R}^n$ are evaluations of $f$ and $g$ at sets of points $\mathbf{x}$ and $\mathbf{y}$ in $\Omega$, respectively, and $W$ is a matrix. The rank of the bilinear form is equal to the rank of $W$, hence the minimal number of required function evaluations is equal to the dimension of that function space.

Accuracy is achieved by defining and minimizing integration error. In a bilinear quadrature, this is a nonlinear optimization problem for $\mathbf{x}, \mathbf{y}$, and $W$ in (1.4), and is solved using a Newton method for an appropriate objective function [CRY99, BGR10, XG10]. In this paper an objective function is developed and demonstrated to yield numerically useful bilinear quadrature rules in a general setting.

Numerical evaluation of inner product integrals has been studied in [BD71, McG79, Gri80, BGR10, Che12] and as “bilinear quadrature” in [LZ87, Kno07]. This paper borrows some of the framework from these past works but develops and utilizes a different optimization procedure to produce quadrature rules.

## 2 Theory

### 2.1 Abstract formulation

In this section the problem of evaluating a general continuous bilinear form on a pair of Banach spaces is considered. Results are given in great generality so that they apply to any continuous bilinear forms. Later, these results are applied to useful special cases such as the $L^2$ and $H^1$ inner products.

**Definition 2.1.** Let $\mathcal{F}$ and $\mathcal{G}$ be real Banach spaces. Then a *bilinear quadrature* of order $(m, n)$ on $\mathcal{F} \times \mathcal{G}$ is a bilinear form $Q$ defined by linear maps $L_1 : \mathcal{F} \to \mathbb{R}^m$ and $L_2 : \mathcal{G} \to \mathbb{R}^n$ and a bilinear map $B : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$, such that, for each $f \in \mathcal{F}$ and $g \in \mathcal{G}$,

$$Q(f, g) = B(L_1 f, L_2 g).$$

**Definition 2.2.** Let $\mathcal{F}, \mathcal{G}$ be real Banach spaces with a continuous bilinear form $\langle \cdot, \cdot \rangle : \mathcal{F} \times \mathcal{G} \to \mathbb{R}$. Finite-dimensional subspaces $\mathcal{F}_0 \subset \mathcal{F}$ and $\mathcal{G}_0 \subset \mathcal{G}$ are a *dual pair* if

$$\forall f \in \mathcal{F}_0 \setminus \{0\}, \exists g \in \mathcal{G}_0 \text{ such that } \langle f, g\rangle \neq 0,$$
$$\forall g \in \mathcal{G}_0 \setminus \{0\}, \exists f \in \mathcal{F}_0 \text{ such that } \langle f, g\rangle \neq 0.$$

If $\mathcal{F}_0, \mathcal{G}_0$ are a dual pair then $\dim(\mathcal{F}_0) = \dim(\mathcal{G}_0)$.

**Definition 2.3.** Let $\mathcal{F}, \mathcal{G}$ be real Banach spaces with a continuous bilinear form $\langle \cdot, \cdot \rangle : \mathcal{F} \times \mathcal{G} \to \mathbb{R}$, and let $\mathcal{F}_0 \subset \mathcal{F}$ and $\mathcal{G}_0 \subset \mathcal{G}$ be a dual pair. A bilinear quadrature $Q$ on $\mathcal{F} \times \mathcal{G}$ is *exact with respect to* $\mathcal{F}_0 \times \mathcal{G}_0$ if

$$\langle f, g\rangle = Q(f, g) \text{ for every } f \in \mathcal{F}_0, g \in \mathcal{G}_0.$$

Such a bilinear quadrature evaluates the bilinear form on $\mathcal{F}_0 \times \mathcal{G}_0$ exactly. We have the diagram

$$\begin{tikzcd}
\mathcal{F}_0 \times \mathcal{G}_0 \arrow[r, "{(L_1, L_2)}"] \arrow[dr, "{\langle \cdot, \cdot \rangle}"'] & \mathbb{R}^m \times \mathbb{R}^n \arrow[d, "{B(\cdot,\cdot)}"] \\
 & \mathbb{R}
\end{tikzcd}$$

If the parent spaces $\mathcal{F}, \mathcal{G}$ are implied, we will abuse notation by referring to an exact bilinear quadrature on $\mathcal{F}_0 \times \mathcal{G}_0$.

*Remark.* If $\mathcal{F}, \mathcal{G}$ are infinite-dimensional and $Q$ is exact on $\mathcal{F}_0 \times \mathcal{G}_0$, then

$$\sup_{f \in \mathcal{F}, g \in \mathcal{G}} |Q(f,g) - \langle f, g \rangle| = \infty,$$

so a bilinear quadrature can only be accurate on finite-dimensional subspaces.

**Lemma 2.4.** *Let $\mathcal{F}_0 \subset \mathcal{F}$ and $\mathcal{G}_0 \subset \mathcal{G}$ be a dual pair, and let $L_1 : \mathcal{F} \to \mathbb{R}^m$, $L_2 : \mathcal{G} \to \mathbb{R}^n$ be linear. Then there exists bilinear $B : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$ such that the map $Q(f,g) = B(L_1 f, L_2 g)$ is an exact quadrature on $\mathcal{F}_0 \times \mathcal{G}_0$ if and only if $L_1|_{\mathcal{F}_0}$ and $L_2|_{\mathcal{G}_0}$ are both injective.*

*Proof.* Suppose $B$ exists. If $f, \tilde{f} \in \mathcal{F}_0$ are distinct then there exists $g \in \mathcal{G}_0$ such that

$$0 \neq \langle f - \tilde{f}, g \rangle = Q(f - \tilde{f}, g) = B(L_1(f - \tilde{f}), L_2 g),$$

so $L_1 f \neq L_1 \tilde{f}$ and $L_1|_{\mathcal{F}_0}$ is injective. Similarly for $L_2|_{\mathcal{G}_0}$.

Suppose $L_1|_{\mathcal{F}_0}$ and $L_2|_{\mathcal{G}_0}$ are injective. Their Moore-Penrose pseudoinverses $(L_1|_{\mathcal{F}_0})^+$ and $(L_2|_{\mathcal{G}_0})^+$ left-invert $L_1$ and $L_2$, respectively. Define a bilinear map on $\mathbb{R}^m \times \mathbb{R}^n$ by

$$B(x,y) = \left\langle (L_1|_{\mathcal{F}_0})^+ x, (L_2|_{\mathcal{G}_0})^+ y \right\rangle.$$

Then, for all $f \in \mathcal{F}_0, g \in \mathcal{G}_0$,

$$\begin{aligned} B(L_1 f, L_2 g) &= \left\langle (L_1|_{\mathcal{F}_0})^+ \, L_1|_{\mathcal{F}_0} f, (L_2|_{\mathcal{G}_0})^+ \, L_2|_{\mathcal{G}_0} g \right\rangle \\ &= \langle f, g \rangle. \end{aligned}$$

□

From Lemma 2.4 a necessary condition for an exact bilinear quadrature is that $m \geq \dim \mathcal{F}_0, n \geq \dim \mathcal{G}_0$. Minimal order is achieved when $m = \dim \mathcal{F}_0, n = \dim \mathcal{G}_0$ and $B(x,y)$ is uniquely given by

$$B(x,y) = \left\langle (L_1|_{\mathcal{F}_0})^{-1} x, (L_2|_{\mathcal{G}_0})^{-1} y \right\rangle.$$

Exact bilinear quadratures are not unique, as there are many possible linear maps $L_1, L_2$. Furthermore, $B$ may not be unique, since if $n > \dim(\mathcal{F}_0)$, then $L_1|_{\mathcal{F}_0}$ has infinitely many left inverses. Therefore, a method is needed to choose among the infinitely many bilinear quadratures. One metric of quality is that, in addition to its exactness on $\mathcal{F}_0 \times \mathcal{G}_0$, the bilinear quadrature also approximates $\langle f, g \rangle$ for some set of $g$'s outside of $\mathcal{G}_0$.

**Definition 2.5.** Let $\mathcal{F}_0 \subset \mathcal{F}$ and $\mathcal{G}_0 \subset \mathcal{G}$ be a dual pair, and let $\mathcal{G}_1 \subset \mathcal{G}$ be another finite-dimensional subspace such that

$$\mathcal{G}_1 \subset \mathcal{F}_0^\perp := \{ g \in G : \langle f, g \rangle = 0 \text{ for all } f \in \mathcal{F}_0 \}.$$

Let $\mathcal{Q}$ be a set of bilinear quadratures exact on $\mathcal{F}_0 \times \mathcal{G}_0$. Then $Q \in \mathcal{Q}$ is called *minimal on* $\mathcal{G}_1$ if

$$Q = \arg\min_{\tilde{Q} \in \mathcal{Q}} \sigma(\tilde{Q}; \mathcal{F}_0, \mathcal{G}_1), \tag{2.1}$$

where

$$\sigma(\tilde{Q};\mathcal{F}_0,\mathcal{G}_1) := \max_{\substack{0\neq g\in\mathcal{G}_1\\ 0\neq f\in\mathcal{F}_0}} \frac{|\tilde{Q}(f,g)|}{\|f\|_{\mathcal{F}}\|g\|_{\mathcal{G}}}.$$

If $Q$ is minimal on $\mathcal{G}_1$, then it approximates the pairing of $\mathcal{F}_0$ and $\mathcal{G}_0 \oplus \mathcal{G}_1$. Precisely, if $f \in \mathcal{F}_0, g \in \mathcal{G}_0 \oplus \mathcal{G}_1$, and we write $g = g_0 + g_1$ with $g_i \in \mathcal{G}_i$, then

$$|Q(f,g) - \langle f,g\rangle| = |Q(f,g_1)| \leq \sigma(Q;\mathcal{F}_0,\mathcal{G}_1)\|f\|_{\mathcal{F}}\|g_1\|_{\mathcal{G}}. \tag{2.2}$$

Thus, minimizing $\sigma(Q;\mathcal{F}_0,\mathcal{G}_1)$ will improve the approximation.

One important special case for bilinear quadratures is the symmetric case, which is when $F = G$ is an inner product space. In this case, a bilinear quadrature computes an orthogonal projection.

**Definition 2.6.** Let $\mathcal{F}_0$ and $\mathcal{G}_0$ be a dual pair in an inner product space. Let $\{f_i\}$ be an orthonormal basis for $\mathcal{F}_0$. Given a bilinear quadrature $Q$ exact on $\mathcal{F}_0 \times \mathcal{G}_0$, the *approximate orthogonal projection* onto $\mathcal{F}_0$ arising from $Q$ is the linear map $P_Q$ given by

$$P_Q(g) = \sum_i Q(f_i,g)f_i.$$

An error estimate for orthogonal projections similar to (2.2) is given later in Theorem 2.7.

## 2.2 Integral formulation

In this section, the bilinear quadrature framework is applied to the evaluation of Sobolev inner products on function spaces. Let $\Omega \subset \mathbb{R}^d$ be a bounded domain, and let $\mathcal{F} = \mathcal{G} = C^r(\overline{\Omega})$, $r$ a non-negative integer, equipped with a Sobolev inner product

$$\langle f,g\rangle_{H^s} = \sum_{|\alpha|\leq s} \langle D^\alpha f, D^\alpha g\rangle_{L^2(\Omega)}$$

for $s \leq r$.

Choose a dual pair $\mathcal{F}_0, \mathcal{G}_0$ in $\mathcal{F}$. Exactness on $\mathcal{F}_0 \times \mathcal{G}_0$ requires linear maps $L_1 : C^r(\overline{\Omega}) \to \mathbb{R}^m$, $L_2 : C^r(\overline{\Omega}) \to \mathbb{R}^n$, and bilinear form $B : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$ so that for every $f \in \mathcal{F}_0, g \in \mathcal{G}_0$, $B(L_1 f, L_2 g) = \langle f,g\rangle_{H^s}$.

Appropriate linear maps $L_1, L_2$ are pointwise evaluations at particular points in $\Omega$. Thus, for the points $\mathbf{x} = (x_1,\dots,x_m) \in \Omega^m$, $\mathbf{y} = (y_1,\dots,y_n) \in \Omega^n$, define

$$L_1 f := f(\mathbf{x}) = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_m)\end{bmatrix}, \quad L_2 g := g(\mathbf{y}) = \begin{bmatrix} g(y_1) \\ \vdots \\ g(y_n)\end{bmatrix}.$$

Given bases $\beta = \{f_1,\dots,f_k\}$ for $\mathcal{F}_0$ and $\{g_1,\dots,g_k\}$ for $\mathcal{G}_0$, let $M \in \mathbb{R}^{k\times k}$ be the Gram matrix with entries

$$M_{ij} = \langle f_i, g_j\rangle_{H^s}.$$

Since $\mathcal{F}_0, \mathcal{G}_0$ are a dual pair, $M$ is invertible. Define matrix functions

$$F(\mathbf{x}) := \begin{bmatrix} \mathcal{F}_1(x_1) & \dots & f_k(x_1) \\ \vdots & & \vdots \\ \mathcal{F}_1(x_m) & \dots & f_k(x_m)\end{bmatrix}, \quad G(\mathbf{y}) := \begin{bmatrix} \mathcal{G}_1(y_1) & \dots & g_k(y_1) \\ \vdots & & \vdots \\ \mathcal{G}_1(y_n) & \dots & g_k(y_n)\end{bmatrix}.$$

To make $L_1$ and $L_2$ are injective, choose $\mathbf{x},\mathbf{y}$, such that $F(\mathbf{x})$ and $G(\mathbf{y})$ have full column rank. If $B(v,w) = v^*Ww$ for all $v,w$ for an $m \times n$ matrix $W$, then the bilinear quadrature is exact if and only if

$$F(\mathbf{x})^*WG(\mathbf{y}) = M. \tag{2.3}$$

Therefore a bilinear quadrature rule

$$Q(f,g) = f(\mathbf{x})^* W g(\mathbf{y}) \tag{2.4}$$

evaluates $\langle f, g\rangle_{H^s}$ exactly for any $f \in \mathcal{F}_0, g \in \mathcal{G}_0$. The corresponding approximate orthogonal projection onto $\mathcal{F}_0$ is

$$P_Q(g) = \sum_{i=1}^{k} \left[f_i(\mathbf{x})^* W g(\mathbf{y})\right] f_i.$$

In the basis $\beta$, the approximate projection is computed by

$$[P_Q(g)]_\beta = F(\mathbf{x})^* W g(\mathbf{y}) \in \mathbb{R}^k. \tag{2.5}$$

Good values for the matrix $W$ and evaluation points $\mathbf{x}, \mathbf{y}$ must be determined. Without loss of generality, suppose that the bases $\{f_i\}$ and $\{g_j\}$ are $H^s$-orthonormal in $C^r(\overline{\Omega})$. Select finite-dimensional $\mathcal{G}_1 \subset C^r(\overline{\Omega})$ for the minimization (2.1) and define the feasible set $\mathcal{Q}$ to be all quadratures of the form (2.4) satisfying (2.3). If $\{\gamma_1, \ldots, \gamma_p\}$ is an orthonormal basis for $\mathcal{G}_1$, define

$$\Gamma(\mathbf{x}) := \begin{bmatrix} \gamma_1(x_1) & \ldots & \gamma_p(x_1) \\ \vdots & & \vdots \\ \gamma_1(x_n) & \ldots & \gamma_p(x_n) \end{bmatrix} \in \mathbb{R}^{n\times p}.$$

Then (2.1) can be reformulated as

$$\begin{aligned} \min_{Q\in\mathcal{Q}} \sigma(Q; \mathcal{F}_0, \mathcal{G}_1) &= \min_{Q\in\mathcal{Q}} \max_{\substack{g\in\mathcal{G}_1, \|g\|_G=1 \\ f\in\mathcal{F}_0, \|f\|_F=1}} |Q(f,g)| \\ &= \min_{\mathbf{x},\mathbf{y},W} \max_{\substack{a,b\in\mathbb{R}^k \\ \|a\|_2=\|b\|_2=1}} |b^* F(\mathbf{x})^* W \Gamma(\mathbf{y}) a| \\ &= \min_{\mathbf{x},\mathbf{y},W} \sigma_1\left(F(\mathbf{x})^* W \Gamma(\mathbf{y})\right) \text{ subject to } F(\mathbf{x})^* W G(\mathbf{y}) = M, \end{aligned} \tag{2.6}$$

where $\sigma_1(A)$ is the leading singular value of a matrix $A$. Minimization (2.6) is independent of $\mathbf{x}$, since by (2.3) $F(\mathbf{x})^* W = ML$, where $L$ is a left inverse of $G(\mathbf{y})$. Therefore $\mathbf{x}$ is chosen by performing a similar minimization on the left, setting an orthonormal basis $\{\lambda_i\}$ for a space $\mathcal{F}_1 \subset \mathcal{G}_0^\perp$, defining the corresponding matrix function $\Lambda(\mathbf{x})$, and minimizing

$$\min_{\mathbf{x},\mathbf{y},W} \sigma_1\left(\Lambda(\mathbf{x})^* W G(\mathbf{y})\right) \text{ subject to } F(\mathbf{x})^* W G(\mathbf{y}) = M, \tag{2.7}$$

where similarly the dependence of (2.7) on $\mathbf{y}$ may be dropped since $WG(\mathbf{y})$ is equal to $L^* M$, where $L$ is a left inverse of $F(\mathbf{x})$.

In the symmetric case $\mathcal{F}_0 = \mathcal{G}_0$, $M = I$, $\mathcal{F}_1 = \mathcal{G}_1$, and $m = n = k$ with $\mathbf{x} = \mathbf{y}$, minimizations (2.6) and (2.7) are equivalent and simplify to

$$\min_{\mathbf{x}} \sigma_1(F(\mathbf{x})^{-1}\Gamma(\mathbf{x})). \tag{2.8}$$

In subsequent sections special attention is given to the symmetric case because it is used for evaluating orthogonal projections.

## 2.3 Error estimates

In this section, upper bounds on several error quantities in computing an approximate orthogonal projection of the form (2.5) are estimated.

**Theorem 2.7** (Euclidean norm error estimate)**.** *Let $\mathcal{F}_0, \mathcal{G}_0$ be a dual pair in an inner product space $\mathcal{F}$ and $Q$ a bilinear quadrature of the form (2.4) that is exact on $\mathcal{F}_0 \times \mathcal{G}_0$. Let $P_Q$ be the approximate orthogonal projection onto $\mathcal{F}_0$ arising from $Q$ with coordinate* *representation* (2.5)*. If $P$ is the exact orthogonal projection operator onto $\mathcal{F}_0$, $\mathcal{G}_1 \subset \mathcal{F}_0^\perp$, and $g = g_0 + g_1 \in \mathcal{G}_0 \oplus \mathcal{G}_1$ such that $g_i \in \mathcal{G}_i$,*

$$\|[P_Q(g) - P(g)]_\beta\|_2 \le \sigma(Q; \mathcal{F}_0, \mathcal{G}_1)\|g_1\|, \tag{2.9}$$

*where $\|\cdot\|_2$ is the Euclidean norm.*

*Proof.* This is essentially the same as (2.1). Since $\langle f_i, g\rangle = Q(f_i, g_0)$, then

$$\begin{aligned}\|[P_Q(g) - P(g)]_\beta\|_2 &= \left(\sum_{i=1}^k |Q(f_i, g) - \langle f_i, g\rangle|^2\right)^{1/2}\\ &= \left(\sum_{i=1}^k |Q(f_i, g) - Q(f_i, g_0)|^2\right)^{1/2}\\ &= \left(\sum_{i=1}^k |Q(f_i, g_1)|^2\right)^{1/2}\\ &= \max_{\alpha\neq 0} \frac{1}{\|\alpha\|_2}\sum_{i=1}^k \alpha_i Q(f_i, g_1),\end{aligned}$$

where $\alpha = (\alpha_i) \in \mathbb{R}^k$. Each $f \in \mathcal{F}_0$ can be written as $f = \sum_i \alpha_i f_i$, so

$$\begin{aligned}\max_{\alpha\neq 0} \frac{1}{\|\alpha\|_2}\sum_{i=1}^k \alpha_i Q(f_i, g_1) &= \max_{0\neq f\in\mathcal{F}_0} \frac{Q(f, g_1)}{\|f\|}\\ &\le \sigma(Q; \mathcal{F}_0, \mathcal{G}_1)\|g_1\|.\end{aligned}$$

□

Theorem 2.7 provides an error bound for an approximate orthogonal projection when the projected function $g$ is in $\mathcal{G}_0 \oplus \mathcal{G}_1$. If $\mathcal{F}_0$ is a space of polynomials, then it is also useful to obtain an error estimate that depends on the regularity of $g$.

**Theorem 2.8** (Uniform norm error estimates for polynomials)**.** *Let $\mathcal{F} = C(\overline{\Omega})$ with $\Omega \subset \mathbb{R}^d$ a bounded, convex domain, equipped with the $L^2$ inner product. Let $\mathcal{F}_0$ be the set of multivariate polynomials of degree at most $n$ with an orthonormal basis $\beta = \{f_i\}$, let $P : \mathcal{F} \to \mathcal{F}$ be the orthogonal projection onto $\mathcal{F}_0$, and suppose $P_Q$ is an approximate orthogonal projection onto $\mathcal{F}_0$ with coordinate representation* (2.5)*. There exist a constant $C > 0$ such that for every $g \in C^{n+1}(\overline{\Omega})$, then*

$$\|[Pg - P_Q g]_\beta\|_\infty \le C\|D^{n+1}g\|_{L^\infty}, \tag{2.10}$$

*where*

$$\|D^{n+1}g\|_{L^\infty} := \sum_{|\alpha|=n+1} \max_{x\in\Omega} |D^\alpha g(x)|.$$

*Proof.* Using (2.5) and the exactness of $P_Q$ on $\mathcal{F}_0$, then writing $g = Pg + (I-P)g = g_0 + g_1$, we have

$$\begin{aligned}\|[Pg - P_Q g]_\beta\|_\infty &= \|F^* W g_1(\mathbf{x})\|_\infty\\ &\le \|F^*W\|_{\infty\to\infty}\|(I-P)g\|_{C^0}\\ &\le \|F^*W\|_{\infty\to\infty}\|(I-P)(g-q)\|_{C^0},\end{aligned}$$

where $q$ is any element of $\mathcal{F}_0$ and $\|\cdot\|_{\infty\to\infty}$ is the induced matrix $\infty$ norm. Then

$$\|[Pg - P_Q g]_\beta\|_\infty \le \|F^*W\|_{\infty\to\infty}(1 + \|P\|_{C^0\to C^0})\|g-q\|_{C^0},$$

where the $C^0$ operator norm of $P$ is given by

$$\|P\|_{C^0 \to C^0} = \max_{x \in \Omega} \int_{\Omega} \Big| \sum_i f_i(t) f_i(x) \Big| \, dt.$$

By the Deny-Lions/Bramble-Hilbert lemma [EG04], for all $g \in C^{n+1}(\Omega)$ there exists a constant $C_{BH} > 0$ (dependent on $n$ and $\Omega$) such that

$$\inf_{q \in \mathcal{F}_0} \|g - q\|_{C^0} \leq C_{BH} \|D^{n+1} g\|_{L^\infty},$$

which combined with the previous inequality yields the desired result with

$$C = \|F^* W\|_{\infty \to \infty} (1 + \|P\|_{C^0 \to C^0}) C_{BH}.$$

□

In the presence of round-off error in function evaluation, the conditioning of an approximate orthogonal projection is also important to quantify.

**Theorem 2.9.** *Let $P_Q$ be an approximate orthogonal projection of the form (2.5). If $\delta g(\mathbf{y})$ is the absolute error in computing $g(\mathbf{y})$ and $\delta P_Q(g)$ is the resulting projection absolute error, then with respect to a vector norm $\|\cdot\|$,*

$$\frac{\|[\delta P_Q(g)]_\beta\|}{\|[P_Q(g)]_\beta\|} \leq \kappa \frac{\|\delta g(\mathbf{y})\|}{\|g(\mathbf{y})\|},$$

*where $\kappa = \kappa(F^*(\mathbf{x}) W)$ is the matrix condition number with respect to $\|\cdot\|$.*

## 2.4 Classical and bilinear quadratures on univariate polynomials

In this section we review Gaussian quadratures and show they are a special case of a bilinear quadrature in one dimension. We then propose a way to generalize to quadratures evaluating inner products of polynomials on multidimensional domains.

**Definition 2.10.** Let $\Omega \subset \mathbb{R}^d$ be a connected domain. A *classical quadrature* $q$ of order $n$ on $\Omega$ is a linear functional defined by a set $\mathbf{x} = (x_1, \ldots, x_n)$, $x_i \in \Omega$, called the *nodes*, and a vector $\mathbf{w} \in \mathbb{R}^n$, whose components are called the *weights*, such that for any $f \in C(\Omega)$,

$$q(f) = \mathbf{w}^* f(\mathbf{x}) = \sum_{i=1}^n w_i f(x_i).$$

Furthermore, if $\mathcal{F}_0$ is a subspace of $C(\Omega)$ and $\mu$ is a Borel measure, then $q$ is said to be exact on $\mathcal{F}_0$ if

$$q(f) = \int_\Omega f \, d\mu$$

for all $f \in \mathcal{F}_0$.

Let $\mathbb{P}_n$ be the space of univariate polynomials of degree up to $n$, $I$ an open interval, and $\mu$ a finite absolutely continuous Borel measure on $I$.

**Definition 2.11.** Suppose $\mathbb{P}_{2n-1}$ is $\mu$-integrable on $I$. Then a *Gaussian quadrature* of order $n$ on $I$ is a classical quadrature of order $n$ on $I$ that is exact on $\mathbb{P}_{2n-1}$ with respect to $\mu$.

The advantages and disadvantages of the theory of quadratures for polynomials are rooted in existence and uniqueness result for Gaussian quadratures.

**Theorem 2.12.** *Suppose $\mathbb{P}_{2n-1}$ is $\mu$-integrable on $I$, and let $\{\phi_k\}$ denote any set of $L^2(I, \mu)$-orthonormal polynomials such that $\deg(\phi_k) = k$. Then the following sets are equal:*

1. *The zeros of $\phi_n$.*

2. *The eigenvalues of the symmetric bilinear form on $\mathbb{P}_{n-1}$ given by*

$$B(f,g) := \int_I x f(x) g(x)\, d\mu = \langle x f(x), g(x) \rangle_{L^2(I,\mu)}\,.$$

3. *The nodes $\{x_i\}$ of a Gaussian quadrature of order $n$ on $I$.*

*Proof.* ($1 \iff 2$) Since $B(\cdot,\cdot)$ is symmetric it is diagonalizable with $n$ real eigenvalues $\{\lambda_i\}$. If a polynomial $\psi_i(x)$ is an eigenvector for $\lambda_i$, then for $0 \le j \le n-1$,

$$\langle x\psi_i(x), \phi_j(x)\rangle = \lambda_i \langle \psi_i(x), \phi_j(x)\rangle \implies \langle (x-\lambda_i)\psi_i(x), \phi_j(x)\rangle = 0.$$

Since $(x-\lambda_i)\psi_i(x) \in \mathbb{P}_n$ for each $i$, and the only polynomials in $\mathbb{P}_n$ that are orthogonal to each of $\phi_0, \dots, \phi_{n-1}$ are multiples of $\phi_n$, then each $(x-\lambda_i)$ is a factor of $\phi_n(x)$. Thus $\phi_n(x)$ is a multiple of $(x-\lambda_1)\dots(x-\lambda_n)$ and its zeros are the eigenvalues of $B(\cdot,\cdot)$.

($2 \iff 3$) Suppose a Gaussian quadrature with weights $\{w_i\}$ and nodes $\{x_i\}$ exists. With respect to the basis of orthonormal polynomials $\{\phi_k\}$, the bilinear form $B(\cdot,\cdot)$ has a symmetric matrix represention $B$ with entries given by

$$B_{ij} = \int_I x\phi_i(x)\phi_j(x)\, d\mu = \sum_{k=1}^n w_k x_k \phi_i(x_k)\phi_j(x_k).$$

If

$$u_k = \begin{bmatrix} \sqrt{w_k}\phi_0(x_k) \\ \vdots \\ \sqrt{w_k}\phi_{n-1}(x_k) \end{bmatrix}, X = \begin{bmatrix} x_1 & & \\ & \ddots & \\ & & x_n \end{bmatrix},$$

then

$$B = \sum_{k=1}^n x_k u_k u_k^* = UXU^*. \tag{2.11}$$

Since $\delta_{ij} = \sum_{k=1}^n w_k\phi_i(x_k)\phi_j(x_k)$, then $I = UU^*$ and $U$ is a unitary matrix. Then (2.11) is the unitary diagonalization of the symmetric matrix $B$ with eigenvalues given by the $x_k$'s. □

*Remarks.* Theorem 2.12 shows that if a Gaussian quadrature of order $n$ exists, its nodes are the zeros of $\phi_n$. The existence proof is completed by showing the weights exist and satisfy

$$1/w_i = \sum_{j=0}^n (\phi_j(x_i))^2.$$

Therefore, taking the square root $\sqrt{w_k}$ is legitimate [DR84]. Theorem 2.12 also provides an efficient method to construct these quadratures. The matrix $B$ in (2.11) is tridiagonal, so its eigenvalues can be calculated quickly, even for very large $n$ [GW69].

Gaussian quadrature is optimal for integrating polynomials on an interval, but does not extend readily to higher-dimensional domains. The zeros of a multivariate polynomial are generally not isolated (consider $f(x,y) = xy$) so they cannot all be used as nodes of a classical quadrature. Additionally, the connection between nodes and eigenvalues no longer holds since the eigenvalues are only scalars (the connection extends to two-dimensional domains with complex eigenvalues [VR14]).

Bilinear quadratures make sense in any dimension, yet contain Gaussian quadrature as a special case. Consider a classical quadrature with nodes $\mathbf{x} = \{x_i\}$ and weights $\mathbf{w} = \{w_i\}$. If a function $h(x) = f(x)g(x)$ with $f, g$ belonging to function spaces $\mathcal{F}, \mathcal{G}$ respectively, then the classical quadrature $q$ evaluated on $h$ is the same as a bilinear quadrature $Q$ on $f, g$ given by

$$Q(f,g) = f(\mathbf{x})^* \begin{bmatrix} w_1 & & \\ & \ddots & \\ & & w_n \end{bmatrix} g(\mathbf{x}) = \mathbf{w}^* h(\mathbf{x}) = q(h).$$

The matrix $W$ is diagonal with entries given by the weights of the classical quadrature. Thus for a general bilinear quadrature of the form (2.4) the entries of $W$ can be viewed as analogues of the weights.

**Theorem 2.13.** *The nodes of a Gaussian quadrature of order $n$ are the same as the points $\mathbf{x}$ in the unique bilinear quadrature of order $(n,n)$ on $\mathbb{P}_{n-1} \times \mathbb{P}_{n-1}$ that is minimal on* $\operatorname{span}\{\phi_n\}$*, where $\phi_n$ is the orthonormal polynomial of degree $n$. Furthermore, the matrix $W$ in (2.4) is a diagonal matrix whose diagonal entries are the weights of the Gaussian quadrature.*

*Proof.* Let $\phi_0, \dots, \phi_{n-1}$ be the orthonormal polynomials up to degree $n-1$ such that $\deg \phi_k = k$, $\mathbf{x} = (x_1, \dots, x_n) \in \Omega^n$, and define

$$\Phi(\mathbf{x}) = \begin{bmatrix} \phi_0(x_1) & \dots & \phi_{n-1}(x_1) \\ \vdots & & \vdots \\ \phi_0(x_n) & \dots & \phi_{n-1}(x_n) \end{bmatrix}.$$

Then the minimization problem (2.8) becomes

$$\min_{\mathbf{x}\in\Omega^n} \sigma_1 \left( \Phi(\mathbf{x})^{-1} \begin{bmatrix} \phi_n(x_1) \\ \vdots \\ \phi_n(x_n) \end{bmatrix} \right).$$

This is uniquely minimized (up to reordering of the $x_i$'s) when $\mathbf{x}$ is the set of zeros of $\phi_n$ in which case $\sigma_1 = 0$. The corresponding bilinear quadrature $Q$ exactly evaluates products where one polynomial has degree $n-1$ and the other has degree $n$. Then $Q$ has the same evaluation points as a bilinear quadrature formed from the Gaussian quadrature. Since $W$ is unique, then it must be equal to the diagonal matrix with entries given by the weights of the Gaussian quadrature. □

The above result suggests that a good way to accurately compute inner products of polynomials on a multidimensional domain is to utilize a symmetric bilinear quadrature that is exact on $\mathbb{P}_n \times \mathbb{P}_n$ and minimal on $\mathbb{P}_{n+1} \cap \mathbb{P}_n^{\perp}$. Just as Gaussian quadratures accurately integrate nearly polynomial functions accurately, bilinear quadratures constructed in the above manner are expected to evaluate inner products of nearly polynomial functions accurately. Numerical results for these quadrature are shown in section 3.

## 2.5 Classical and bilinear quadratures on trigonometric polynomials

For the space of trigonometric polynomials

$$T_{n-1} = \operatorname{span}\{1, \sin x, \dots, \sin{(n-1)x}, \cos x, \dots, \cos{(n-1)x}\},$$

it is known that the $(n+1)$-point trapezoidal rule

$$\mathrm{Tra}(p) := \frac{\pi}{n}p(0) + \frac{\pi}{n}p(2\pi) + \frac{2\pi}{n}\sum_{j=1}^{n-1} p\left(\frac{2\pi j}{n}\right)$$

is exact for integrating all $p \in T_{n-1}$ over the interval $[0, 2\pi]$. Since $T_{n-1}$ is a rotationally-invariant function space on the circle $\mathbb{R}/2\pi\mathbb{Z}$, the trapezoidal rule yields a family of $n$-point classical quadratures for $T_{n-1}$ given by

$$\int_0^{2\pi} p(x)\,dx = \frac{2\pi}{n}\sum_{j=0}^{n-1} p(x_j) \quad \text{for all } p \in T_{n-1}, \quad x_{j+1} - x_j = \frac{2\pi}{n}. \tag{2.12}$$

When $n$ is odd, the above trapezoidal rule quadrature is a special case of a bilinear quadrature:

**Theorem 2.14.** *Let $n > 0$ be an odd integer. Then the set of classical quadratures on $T_{n-1}$ in (2.12) are equivalent to symmetric bilinear quadratures of order $(n, n)$ on $T_{(n-1)/2} \times T_{(n-1)/2}$ that are minimal on* $\text{span}\{\sin nx, \cos nx\}$.

*Proof.* Set $n = 2k + 1$. Since $T_{n-1}$ is rotationally invariant on the circle, then if $Q$ is a symmetric bilinear quadrature on $T_k \times T_k$ of the form (2.4), $\sigma(Q)$ is invariant under rotations of the evaluation points $\mathbf{x} = (x_1, \dots, x_n)$. Therefore without loss of generality $x_1 = 0, x_j \in [0, 2\pi)$. Define

$$F(\mathbf{x}) = \frac{1}{\sqrt{2\pi}}\begin{bmatrix} 1 & \dots & 1 \\ e^{-ikx_2} & \dots & e^{ikx_2} \\ \vdots & & \vdots \\ e^{-ikx_n} & \dots & e^{ikx_n}\end{bmatrix}, \Gamma(\mathbf{x}) = \frac{1}{\sqrt{2\pi}}\begin{bmatrix} 1 & 1 \\ e^{-i(k+1)x_2} & e^{i(k+1)x_2} \\ \vdots & \vdots \\ e^{-i(k+1)x_n} & e^{i(k+1)x_n}\end{bmatrix},$$

and it suffices to prove that choosing $x_j = 2\pi j/n$ solves the minimization problem (2.1).

If $x_j = 2\pi j/n$, then the first column of $F(\mathbf{x})$ is the second column of $\Gamma(\mathbf{x})$, and the last column of $F(\mathbf{x})$ is the first column of $\Gamma(\mathbf{x})$. Therefore, in this case, $F(\mathbf{x})^{-1}\Gamma(\mathbf{x}) = \begin{bmatrix} e_n & e_1 \end{bmatrix}$, where $e_j$ is the $j$-th standard coordinate vector, hence $\sigma_1(F(\mathbf{x})^{-1}\Gamma(\mathbf{x})) = 1$.

We claim that for any choice of nodes $\mathbf{x} \in [0, 2\pi)^n$ with $x_1 = 0$,

$$\sigma_1(F(\mathbf{x})^{-1}\Gamma(\mathbf{x})) \geq 1. \tag{2.13}$$

If (2.13) is established, then setting $x_j = 2\pi(j-1)/n$ yields a minimal quadrature in $\mathcal{Q}$. To show this, let $\mathbf{u}$ be the first column of $F(\mathbf{x})^{-1}\Gamma(\mathbf{x})$. We will show that $\|\mathbf{u}\|_2 \geq 1$, from which (2.13) follows. The column $\mathbf{u}$ satisfies the equation

$$F(\mathbf{x})\mathbf{u} = \frac{1}{\sqrt{2\pi}}\begin{bmatrix} 1 \\ e^{-i(k+1)x_2} \\ \vdots \\ e^{-i(k+1)x_n}\end{bmatrix},$$

which is equivalent to the Vandermonde system

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & e^{ix_2} & \dots & e^{i(n-1)x_2} \\ \vdots & \vdots & & \vdots \\ 1 & e^{ix_n} & \dots & e^{i(n-1)x_n}\end{bmatrix}\mathbf{u} = \begin{bmatrix} 1 \\ e^{-ix_2} \\ \vdots \\ e^{-ix_n}\end{bmatrix}.$$

Setting $z_j = e^{ix_j}$, then the entries of $\mathbf{u}$ are the coefficients of a degree $n-1$ complex polynomial $p(z)$ such that $p(z_j) = 1/z_j$. Setting $q(z) = zp(z)$, it suffices to find a degree $n$ polynomial $q(z)$ such that $q(0) = 0$ and $q(z_j) = 1$. Such a $q$ is unique and

$$q(z) = 1 - \prod_{j=1}^{n}(1 - z/z_j).$$

Then the leading coefficient of $q$, which is also the leading coefficient of $p$, has absolute value 1, and hence $\|\mathbf{u}\|_2 \geq 1$. □

*Remark.* The trapezoidal rule *uniquely* generates a minimal bilinear quadrature, since in that case $q(z) = \alpha z^n$ for some $|\alpha| = 1$. If $x_j$'s are not equispaced, $q(z)$ has some nonzero lower-order coefficients.

## 2.6 Lobatto quadrature and the non-invertible case

While minimizing the number of evaluation points will reduce the cost of evaluating a quadrature, it may be advantageous to use more points than is optimal in order to improve accuracy. One example is Lobatto quadratures for polynomials of one variable, which use more points than Gaussian quadratures. In this section, we observe Lobatto quadratures are a special case of a bilinear quadrature where extra evaluation points are used, in which case the matrix function $F(\mathbf{x})$ is non-invertible. The formulation of Lobatto-like bilinear quadratures on general domains is given.

**Definition 2.15.** Let $\mathbb{P}_{2n-1}$ be $\mu$-integrable on an interval $I = [a, b]$. Then the corresponding *Lobatto quadrature* is a classical quadrature of order $n + 1$ exact on $\mathbb{P}_{2n-1}$ with respect to $\mu$ such that if $x_0, \dots, x_n$ are the nodes, then $x_0 = a$ and $x_n = b$.

**Theorem 2.16.** *Suppose $\mathbb{P}_{2n-1}$ is $\mu$-integrable on $I$, and let $\{\phi_k\}$ denote the unique set of orthonormal polynomials such that $\deg(\phi_k) = k$. Then there exists a unique Lobatto quadrature of order $n + 1$, and the interior nodes $\{x_i : 1 \le i \le n - 1\}$ are the zeros of $\frac{d}{dx}\phi_n(x)$.*

A Lobatto quadrature of order $n + 1$ corresponds to a symmetric bilinear quadrature that is exact on $\mathbb{P}_{n-1} \times \mathbb{P}_{n-1}$ and minimal on $\mathbb{P}_n$ in which the $W$ matrix is diagonal and the matrix $F(\mathbf{x})$ is given by

$$F(\mathbf{x}) = \begin{bmatrix} \phi_1(a) & \dots & \phi_k(a) \\ \phi_1(x_1) & & \phi_k(x_1) \\ \vdots & & \vdots \\ \phi_1(x_{n-1}) & & \phi_k(x_{n-1}) \\ \phi_1(b) & \dots & \phi_k(b) \end{bmatrix}.$$

Unlike in the Gaussian quadrature case, the matrix $F = F(\mathbf{x})$ is not square, so there exists infinitely many matrices $W$ satisfying (2.3). Therefore, the simplified minimization condition (2.8) cannot be employed, and one must optimize over both the quadrature nodes $\mathbf{x}$ and matrices $W$. In general, suppose $F$ is $m \times k$ and $G$ is $n \times k$, both with full column rank. Then all matrices $W$ satisfying (2.3) are of the form

$$W = (F^*)^+ M G^+ + Y - FF^+ Y G G^+, \tag{2.14}$$

where $Y$ is an arbitrary $m \times n$ matrix. In the symmetric case $F = G$ and $M = I$, a minimal bilinear quadrature is found through the unconstrained minimization

$$\text{Find } Y \in \mathbb{R}^{m\times m} \text{ and } \mathbf{x} \text{ minimizing } \sigma_1\left(F^+\Gamma + F^* Y (I - FF^+)\Gamma\right) \tag{2.15}$$

While computationally more expensive, this optimization procedure can be used to compute symmetric Lobatto-like bilinear quadratures. First fix points $\mathbf{x}_0$ that the bilinear quadrature is required to use, then construct the (typically non-square) matrix function $F(\mathbf{x}_0, \mathbf{x})$, where only the points $\mathbf{x}$ are varying. Then minimize according to (2.15). This procedure is applicable for arbitrary domains $\Omega$, any space of continuous functions, and any inner product on that space.

## 2.7 Change of variables

For a bilinear quadrature computing an $L^2(\Omega)$ inner products, a bilinear quadrature can be cheaply constructed for $L^2(\Phi(\Omega))$ inner products, where $\Phi$ is an affine invertible change of variables. For continuous functions $f_i$ on $\Omega$, set

$$\tilde{f}_i(\Phi(x)) = f_i(x)|\det(D\Phi)|^{-1/2}.$$

Then

$$\langle \tilde{f}_i, \tilde{f}_j \rangle_{L^2(\Phi(\Omega))} = \int_{\Phi(\Omega)} \tilde{f}_i(y)\tilde{f}_j(y)\,dy = \int_{\Omega} f_i(x)f_j(x)\,dx = \langle f_i, f_j \rangle_{L^2(\Omega)}.$$

The Jacobian $D\Phi$ is constant when $\Phi$ is affine, so if the bilinear quadrature on $L^2(\Omega)$ exact on $\mathcal{F}_0 \times \mathcal{G}_0$ is

$$Q(f,g) = f(\mathbf{x})^* W g(\mathbf{y}),$$

a bilinear quadrature on $L^2(\Phi(\Omega))$ for $(\mathcal{F}_0 \circ \Phi^{-1}) \times (\mathcal{G}_0 \circ \Phi^{-1})$ is given by

$$\tilde{Q}(\tilde{f}, \tilde{g}) = \tilde{f}(\Phi(\mathbf{x}))^* \tilde{W} \tilde{g}(\Phi(\mathbf{y})), \quad \tilde{W} = W|\det(D\Phi)|^{-1}. \tag{2.16}$$

For an $H^s$ inner product with $s > 0$, in general a new bilinear quadrature cannot be cheaply constructed under a change of variables. However, when $\Phi(x) = \lambda U x + b$ is affine with $\lambda \in \mathbb{R}$ and $U$ a unitary matrix, a change of variables can still be performed at low cost. Let $W$ be the matrix in a bilinear quadrature of form (2.4) computing $H^1(\Omega)$ inner products. Then write $W = W_0 + W_1$, where $W_0$ is the matrix for a bilinear quadrature that computes $L^2(\Omega)$ inner products. Then a new bilinear quadrature for $H^1(\Phi(\Omega))$ is formed with matrix

$$\tilde{W} = |\lambda|^{-1} W_0 + |\lambda|^{-3} W_1$$

and evaluation points mapped by $\Phi$.

## 3 Computation

In this section, a basic numerical procedure to produce symmetric bilinear quadrature rules is described. Afterward, some numerical examples of bilinear quadrature rules are presented.

### 3.1 Orthogonalization

For a function space $\mathcal{F}_0$, one may initially have a numerical routine to evaluate (up to machine precision) basis functions $\psi_1, \ldots, \psi_k$ for $\mathcal{F}_0$ that are not orthonormal. Assuming that the inner products $\langle \psi_i, \psi_j \rangle = M_{ij}$ can be computed exactly, $F(\mathbf{x})$ is computed from $\Psi(\mathbf{x})$ and Gram matrix $M$ by

1. Compute the lower triangular matrix $L$ in the Cholesky factorization $M = LL^*$.
2. For a given $\mathbf{x}$, perform a lower-triangular solve on the matrix equation $\Psi(\mathbf{x})^* = LZ$.
3. Set $F(\mathbf{x}) = Z^*$.

The same procedure can be used to produce an orthonormal basis for the function space $\mathcal{F}_1$ that the bilinear quadrature is minimized against.

### 3.2 Nonlinear optimization

For the invertible symmetric case we have reduced our problem to the minimization problem (2.6):

$$\text{Find } \mathbf{x} \text{ minimizing } \sigma_1\left(F(\mathbf{x})^{-1}\Gamma(\mathbf{x})\right).$$

This is a nonlinear optimization problem in $d \cdot k$ variables, where $d$ is the dimension of the integration region $\Omega$ and $k = \dim(\mathcal{F}_0)$.

The problem of minimizing the largest singular value of a matrix function $A(\mathbf{x})$ is equivalent to minimizing the largest eigenvalue of the symmetric positive semidefinite matrix

$A^*(\mathbf{x})A(\mathbf{x})$. This type of the eigenvalue optimization problem has been extensively studied in its own right; see [OW95] [SF95].

Often $F(\mathbf{x})$ and $\Gamma(\mathbf{x})$, but not their derivatives, can be accurately computed. Also, the multiplicity of the largest singular values are generally unknown. Consequently, a quasi-Newton method is ideal for the optimization procedure. The objective function is non-convex and typically has multiple local minima, so the optimization procedure is run with many initial guesses. Furthermore, in the presence of many nearby local minima, after each convergent result, the computed points can be perturbed by a small value $\delta$ and the procedure run again with perturbed points as another initial guess. This is repeated until suitable convergence. While this procedure may be expensive, computing a quadrature is typically a one-time cost, after which the quadrature can be used repeatedly for its applications.

In our numerical experiments, we employ a quasi-Newton method with BFGS updates as implemented as part of Matlab's `fminunc` routine [Bro70, Fle70, Gol70, Sha70]. Since $F(\mathbf{x})^{-1}\Gamma(\mathbf{x})$ is a small, dense matrix, its norm is computed by calculating its full SVD. Up to $10^5$ initial random points uniformly distributed across the domain are used, and the procedure is iterated until convergence in double-precision arithmetic.

For our numerical implementation we do not reinforce the constraint that the evaluation points $x_i$ remain in the integration domain $\Omega$. While in general the full constrained minimization problem may be necessary, we have empirically observed that it is not necessary for quadratures on polynomials. This can be explained by observing that the orthogonal polynomials grow rapidly outside of $\Omega$; thus points outside the domain are not expected to be good candidates for the solution to the minimization problem.

*Remarks.* In the case of polynomials it is possible to accurately compute the gradients of $F(\mathbf{x})$ and $\Gamma(\mathbf{x})$, in which case a quasi-Newton method may be unnecessary. The BFGS method has been chosen since it is robust for different function spaces.

## 3.3 Bilinear quadratures on triangular domains

In practical applications one of the most important cases to consider is the $L^2$ product of polynomials on a simplex. For example, in the finite element method one typically solves a two-dimensional PDE locally on polynomials supported on triangular domains. The discretization requires computing a number of inner products. In this section we compute bilinear quadratures that are exact on polynomials on a triangular domain.

Because the space of polynomials is affine-invariant it suffices to find evaluation points for polynomials on a reference triangle. Given a bilinear quadrature on a reference triangle a bilinear quadrature for polynomials on any other triangle can be cheaply obtained using the change of variables formula (2.16). A basis of orthogonal polynomials on the right triangle with vertices $(-1,-1), (-1,1), (1,-1)$ is given by

$$K_{m,n}(x,y) = \left(\frac{1-v}{2}\right)^m P_m\left(\frac{2x+y+1}{1-y}\right) P_n^{2m+1,0}(y), \tag{3.1}$$

where $P_m$ is the $m$th Legendre polynomial and $P_n^{\alpha,\beta}$ is the $n$th Jacobi polynomial with parameters $\alpha, \beta$. These functions can be computed efficiently and stably as in [XG10].

Using this basis, symmetric bilinear quadratures exact for the $L^2$ inner product over this right triangle on $\mathcal{F}_0 = \mathbb{P}_n$ and minimal on $\mathcal{F}_1 = \mathbb{P}_{n+1} \cap \mathbb{P}_n^{\perp}$ were computed. The minimal number of evaluation points were used, in which case the number of points required is

$$k = \dim(\mathbb{P}_n) = \binom{n+2}{2}.$$

In Table 1, for each computed bilinear quadrature rule, the minimized largest singular value $\sigma = \sigma_1(F(\mathbf{x})^{-1}\Gamma(\mathbf{x}))$ is given, along with the $\infty$-norm condition number of the matrix for the approximate orthogonal projection.

| $n$ | $k$ | $\sigma$ | $\kappa_\infty(F^*W)$ |
|---|---|---|---|
| 0 | 1 | 0.00000 | 1.00000e+0 |
| 1 | 3 | 0.14507 | 2.82218e+0 |
| 2 | 6 | 0.30373 | 6.29185e+0 |
| 3 | 10 | 0.47762 | 1.15455e+1 |
| 4 | 15 | 0.65817 | 2.03810e+1 |
| 5 | 21 | 0.78394 | 3.39955e+1 |
| 6 | 28 | 0.87930 | 4.71065e+1 |
| 7 | 36 | 0.95305 | 8.48889e+1 |
| 8 | 45 | 1.05595 | 1.09107e+2 |

Table 1: Numerical results for $k$-point bilinear quadratures on $\mathbb{P}_n \times \mathbb{P}_n$ for $L^2$ on the interior of the reference right triangle.

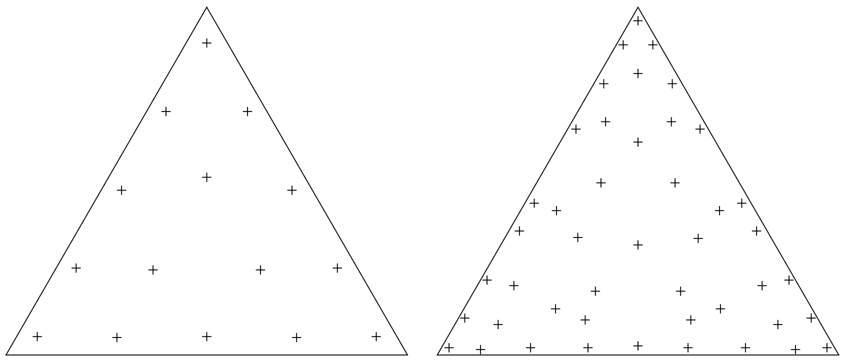


Figure 1: Evaluation points for bilinear quadratures on $\mathbb{P}_n \times \mathbb{P}_n$ for $L^2$ on the interior of an equilateral triangle, for $n = 4, 8$.

In Figure 1, the evaluation points of two bilinear quadrature rules on the equilateral triangle are shown. Notice that the points possess some symmetries. The expectation that quadrature points for polynomials should have some symmetries has been exploited in the past to reduce the complexity of searching for classical quadratures [XG10]. In the quasi-Newton method used to solve (2.8), however, no symmetry conditions were explicitly enforced.

## 3.4 Numerical accuracy of quadratures on triangles

In the section the computed bilinear quadratures on triangles are compared against existing high-order classical quadrature schemes on triangles in the setting of orthogonal projections. Given the space $\mathcal{F}_0 = \mathbb{P}_n$ on $\Omega$ with $L^2$-orthonormal basis $\beta = \{f_i\}$, orthogonal projection operator $P$ onto $\mathcal{F}_0$, and given $g \in C^\infty(\Omega)$, we wish to compute

$$[Pg]_\beta = \begin{bmatrix} \langle f_1, g\rangle_{L^2} \\ \vdots \\ \langle f_k, g\rangle_{L^2} \end{bmatrix}.$$

The column vector $[Pg]_\beta$ can be computed using either an approximate orthogonal projection, or a classical quadrature for each entry $\int_\Omega f_i g$.

Since the approximate orthogonal projection matrix $F^*W$, weights of the classical quadrature, and locations of evaluation points are all precomputed, the flop cost for each method is solely determined by the number of evaluation points needed. The 28-point bilinear

| | $\mathbb{P}'_5$ | $\mathbb{P}'_6$ | $C$ | $TP$ |
|---|---|---|---|---|
| Dunavant | 9.38e-14 | 3.97e-01 | 4.97e-05 | 9.06e-03 |
| Xiao/Gimbutas | 3.29e-15 | 2.73e-01 | 1.91e-05 | 4.74e-03 |
| Bilinear | 3.92e-15 | 3.99e-15 | 6.74e-06 | 1.71e-03 |

Table 2: Average $\ell^2$-norm relative error in computing approximate orthogonal projection coefficients onto $\mathbb{P}_6$ for four different sets of functions using three methods that use 28 function evaluations.

quadrature as shown in Figure 1 was utilized. For comparison we chose two different 28-point classical quadratures, each exact on polynomials of degree up to 11, due to Dunavant [Dun85] and Xiao and Gimbutas [XG10], respectively. These quadratures were computed using the libraries available from [Bur15]. Both classical quadratures were similarly transformed to an equilateral triangle of side length 1.

For our numerical experiments, we draw the projected function $g$ from four different probability distributions of functions, which we denote by $\mathbb{P}'_5, \mathbb{P}'_6, C$, and $TP$.

We define

$$\mathbb{P}'_n := \{g \in \mathbb{P}_n : \|g\|_{L^2} = 1\},$$

with probability measure given by drawing a random vector of coefficients uniformly in $[-1,1]^k$, and then normalizing the coefficients to have $\ell^2$-norm 1, and using those as the Fourier coefficients on the orthonormal polynomials on the triangle.

The set $C$ contains smooth functions with slow decay, and is defined by functions of the form

$$g(x,y) = \frac{1}{1+(a_1x+a_2y)^2},$$

where $a = (a_1, a_2)$ is drawn uniformly from the unit circle.

The set $TP$ contains smooth non-polynomial functions with oscillations, and has elements of the form

$$g(x,y) = e^{a_1x+a_2y}\cos(4b_1x+4b_2y)p(x,y),$$

where parameters $(a_1,a_2)$ and $(b_1,b_2)$ are both drawn uniformly from the unit circle, and $p(x,y)$ is a random element of $\mathbb{P}'_2$ with $L^2$ norm 1 as chosen in the same manner as for the first two cases.

For each randomly chosen function $g$, we computed the column vector $[P_Q g]_\beta$ using the three quadrature methods. The exact value $[Pg]_\beta$ was computed with a 295-point classical quadrature that exactly integrates polynomials up to degree 40, as computed in [XG10]. The $\ell^2$ norm relative error was averaged over $10^4$ randomly generated $g$ for each of the four classes of functions. The resulting average relative errors are shown in Table 2.

On $\mathbb{P}'_5$, all three quadrature rules achieve very high accuracy, with the Dunavant quadrature losing one digit of accuracy and both Xiao/Gimbutas and bilinear quadratures correctly computing the orthogonal projection up to double precision. This is expected since all quadratures are designed to integrate such polynomial functions exactly.

On $\mathbb{P}'_6$, neither classical quadratures are accurate to full precision because both classical quadratures are only capable of exactly integrating polynomials of degree up to 11. Since the bilinear quadrature can exactly integrate $\mathbb{P}_6 \times \mathbb{P}_6$, it has mean error on the order of machine precision.

On the sets $C$ and $TP$, none of the quadratures are accurate to machine precision since none of the functions are polynomials. However, the bilinear quadrature achieves better accuracy than the classical quadratures despite having the same number of evaluation points.

The existing classical quadratures are already very good, integrating non-polynomial functions from $C$ and $TP$ with several digits of accuracy. Additionally, the classical quadrature of Xiao/Gimbutas performs better than the Dunavant quadrature in all four cases. However, the bilinear quadrature was as good or better than the classical quadratures in each case, despite using the same number of evaluations. This result is explained by the fact

| $n$ | $k$ | $\sigma$ | $\kappa_\infty(F^*W)$ |
|---|---|---|---|
| 0 | 1 | 0.00000 | 1.00000e+0 |
| 1 | 3 | 0.67739 | 2.91852e+0 |
| 2 | 6 | 0.79523 | 7.50137e+0 |
| 3 | 10 | 0.92888 | 1.17526e+1 |
| 4 | 15 | 0.97590 | 2.68367e+1 |
| 5 | 21 | 0.99701 | 3.14417e+1 |
| 6 | 28 | 1.00066 | 6.42937e+1 |
| 7 | 36 | 1.00711 | 7.34237e+1 |
| 8 | 45 | 1.00784 | 1.03464e+2 |

Table 3: Numerical results for $k$-point bilinear quadratures on $\mathbb{P}_n \times \mathbb{P}_n$ for $L^2$ on the interior of a square.

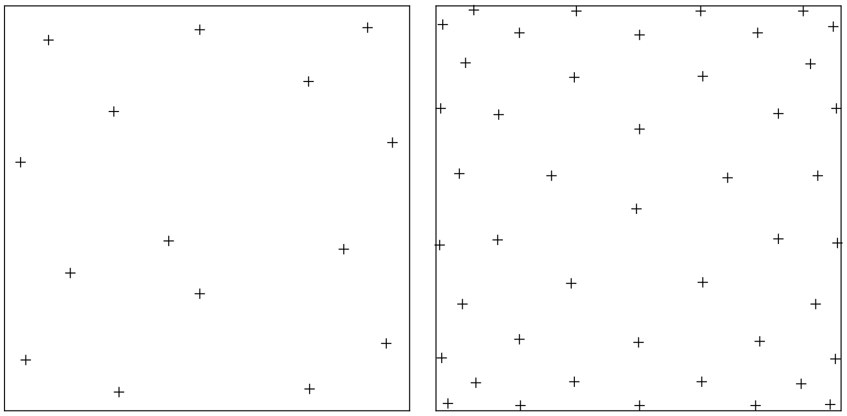


Figure 2: Evaluation points for bilinear quadratures on $\mathbb{P}_n \times \mathbb{P}_n$ for $L^2$ on the interior of a square, for $n = 4, 8$.

that bilinear quadratures are specifically designed for the orthogonal projection problem, while classical quadratures are designed for evaluating a linear functional.

## 3.5 Bilinear quadratures on other domains

In this section bilinear quadratures for $L^2$ inner products of polynomials on the interiors of a square and a circle are computed. We observe that, just as in the case of triangles, minimizing according to (2.8) produces well-behaved evaluation points.

For the case of the square domain $[-1,1]^2$, orthogonal polynomials are $P_n(x)P_m(y)$, where $P_n$ is the $n$th Legendre polynomial. Table 3 shows the minimized leading singular value $\sigma$ and matrix condition number $\kappa_\infty$ for several $k$-point bilinear quadratures on the square. Interestingly, the evaluation points on the square do not appear to obey any symmetries.

*Remark.* One can produce a classical quadrature scheme on the square by simply taking the tensor product of two Gaussian quadratures on an interval. However, this exactly integrates basis functions of the form $x^\alpha y^\beta$ with $0 \le \alpha \le n$, $0 \le \beta \le n$, rather than integrating polynomials whose *total degree* does not exceed some value.

On the unit disk, an orthogonal basis of polynomials is given in polar coordinates by the

| $n$ | $k$ | $\sigma$ | $\kappa_\infty(F^*W)$ |
|---|---|---|---|
| 0 | 1 | 0.00000 | 1.00000e+0 |
| 1 | 3 | 0.67617 | 3.04857e+0 |
| 2 | 6 | 0.79868 | 5.50559e+0 |
| 3 | 10 | 0.89712 | 1.01509e+1 |
| 4 | 15 | 0.94133 | 1.59179e+1 |
| 5 | 21 | 0.97804 | 2.24193e+1 |
| 6 | 28 | 1.00337 | 3.94055e+1 |
| 7 | 36 | 1.02908 | 5.60579e+1 |
| 8 | 45 | 1.07413 | 6.75064e+1 |

Table 4: Numerical results for $k$-point bilinear quadratures on $\mathbb{P}_n \times \mathbb{P}_n$ for $L^2$ on the unit disk.

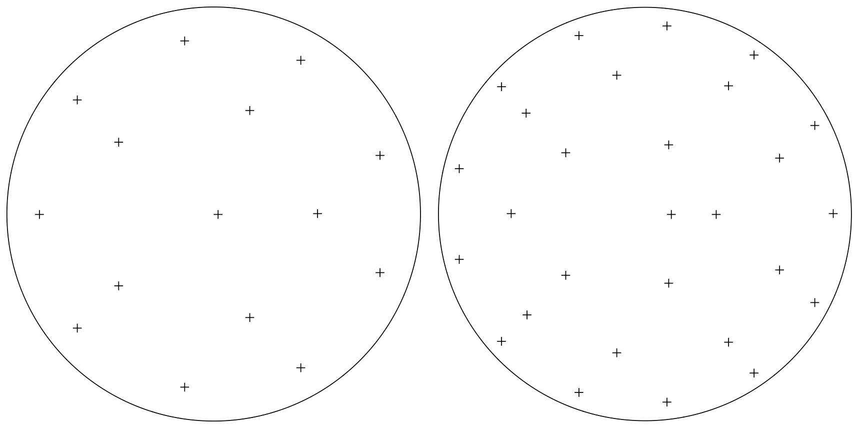


Figure 3: Evaluation points for bilinear quadratures on $\mathbb{P}_n \times \mathbb{P}_n$ for $L^2$ on the unit disk, for $n = 4, 6$.

Zernike polynomials $Z_{m,n}(r,\theta)$, defined by

$$Z_{m,n}(r,\theta) := Q_{m,n}(r)\cos(m\theta), \quad Z_{-m,n}(r,\theta) := Q_{m,n}(r)\sin(m\theta),$$
$$Q_{m,n}(r) := \sum_{k=0}^{(n-m)/2} (-1)^k \binom{n-k}{k}\binom{n-2k}{\frac{n-m}{2}-k} r^{n-2k},$$

where $n \geq m \geq 0$ are integers and $n - m$ is even. Table 4 shows the minimized leading singular value $\sigma$ and matrix condition number $\kappa_\infty$ for several $k$-point bilinear quadratures on the unit disk.

## 3.6 Bilinear quadrature for the Sobolev inner product

In this section we compute bilinear quadratures that evaluate the Sobolev inner product

$$\langle f, g\rangle_{H^1} = \int_\Omega Df(x)\cdot A(x)Dg(x) + f(x)g(x)\,dx,$$

where $A(x)$ is symmetric positive definite on $\Omega$. One advantage of a bilinear quadrature for $H^1$ is that the above integral can be numerically evaluated using only point evaluations of $f, g$ and does not require evaluating any derivatives.

For $\Omega = [-1,1]$, bilinear quadratures for $H^1$ on $\mathbb{P}_n \times \mathbb{P}_n$ and minimal on $\mathbb{P}_{n+1} \cap \mathbb{P}_n^\perp$ were computed for two positive weight functions $A(x) = 1+x^2$ and $A(x) = e^x$. Orthogonalization was performed by starting with the Legendre polynomials and computing the Gram matrix $M$ using a 40-point classical Gaussian quadrature.

| $n$ | $k$ | $\sigma$ | $\kappa_\infty(F^*W)$ |
|---|---|---|---|
| 1 | 2 | 0.00000 | 5.00000e+0 |
| 2 | 3 | 0.00000 | 1.38132e+1 |
| 3 | 4 | 0.00000 | 2.72011e+1 |
| 4 | 5 | 0.00000 | 6.59254e+1 |
| 5 | 6 | 0.00000 | 1.21461e+2 |
| 6 | 7 | 0.00000 | 1.86818e+2 |
| 7 | 8 | 0.00000 | 2.86549e+2 |
| 8 | 9 | 0.00000 | 4.22824e+2 |

Table 5: Numerical results for bilinear quadratures on $\mathbb{P}_n \times \mathbb{P}_n$ for $H^1[-1,1]$ with the weight function $A(x) = 1 + x^2$.

| $n$ | $k$ | $\sigma$ | $\kappa_\infty(F^*W)$ |
|---|---|---|---|
| 1 | 2 | 0.00000 | 4.52560e+0 |
| 2 | 3 | 0.00000 | 1.30446e+1 |
| 3 | 4 | 0.00000 | 2.49183e+1 |
| 4 | 5 | 0.00000 | 5.13338e+1 |
| 5 | 6 | 0.00000 | 9.28987e+1 |
| 6 | 7 | 0.00000 | 1.50063e+2 |
| 7 | 8 | 0.00000 | 2.28284e+2 |
| 8 | 9 | 0.00000 | 3.30651e+2 |

Table 6: Numerical results for bilinear quadratures on $\mathbb{P}_n \times \mathbb{P}_n$ for $H^1[-1,1]$ with the weight function $A(x) = e^x$.

In Tables 5 and 6 the singular value $\sigma$ and condition number $\kappa_\infty$ are shown for the two bilinear quadratures for $H^1$. In all cases, $\sigma$ is zero up to machine precision, since the exact solution to the minimization (2.6) is the roots of the $(n+1)$th-degree $H^1$-orthogonal polynomial, just as for Gaussian quadratures. We observe that the condition number of the approximation projection matrix $F^*W$ is larger than in the $L^2$ case. This can be explained by the fact that small perturbations in the function values can lead to large perturbations in the derivatives.

## 4 Conclusions

A quadrature framework for numerically evaluating a continuous bilinear form on function spaces has been presented, and an optimization procedure for computing such quadratures has been outlined. We have argued that this is the correct approach to numerically evaluating orthogonal projections of functions onto a fixed subspace.

We have also observed that the optimization approach for finding bilinear quadratures does not depend on the ambient dimension, the domain of integration, or the function space to be integrated exactly. Despite this generality, in our numerical experiments we found the resulting quadratures perform well, achieving both efficiency and accuracy.

There are several topics to explore in future work. One is the construction and utilization of bilinear quadratures tailored to specific high-order Galerkin methods. Another is the investigation of the performance of bilinear quadratures for evaluating other (non-Sobolev) bilinear forms. Yet another finding an efficient numerical method for solving the optimization problem (2.15) for the non-invertible case. In that case, a bilinear quadrature is not uniquely determined by its evaluation points, and the optimization problem gains many additional degrees of freedom. Lastly, one could investigate the use of bilinear quadratures for solving integral equations. Such quadratures may prove useful in the Nyström discretization of Fredholm integral operators [Bol72] or boundary integral equations on domains with corners [BRS10].

## 5 Acknowledgements

I would like to thank Ming Gu, Benjamin Harrop-Griffiths, Casey Jao, Per-Olof Persson, and John Strain for comments and suggestions. This work was supported by National Science Foundation under grant DMS-0913695 and the Air Force Office of Scientific Research under grant FA9550-11-1-0242.

## References

[BD71] W. Robert Boland and C.S. Duris, *Product type quadrature formulas*, BIT Numerical Mathematics **11** (1971), no. 2, 139–158.

[BGR10] James Bremer, Zydrunas Gimbutas, and Vladimir Rokhlin, *A nonlinear optimization procedure for generalized Gaussian quadratures*, SIAM J. Sci. Comput. **32** (2010), no. 4, 1761–1788.

[Bol72] W. Robert Boland, *The numerical solution of Fredholm integral equations using product type quadrature formulas*, BIT Numerical Mathematics **12** (1972), no. 1, 5–16.

[Bro70] C. G. Broyden, *The convergence of a class of double-rank minimization algorithms*, IMA Journal of Applied Mathematics **6** (1970), no. 1, 76–90.

[BRS10] James Bremer, Vladimir Rokhlin, and Ian Sammis, *Universal quadratures for boundary integral equations on two-dimensional domains with corners*, Journal of Computational Physics **229** (2010), no. 22, 8259 – 8280.

[Bur15] John Burkardt, *Source codes in Fortran90; source codes in Matlab*, http://people.sc.fsu.edu/~jburkardt/, 2015.

[Che12] Y. Chen, *Inner product quadratures*, ArXiv e-prints (2012).

[CRY99] H. Cheng, V. Rokhlin, and N. Yarvin, *Nonlinear optimization, quadrature, and interpolation*, SIAM J. Optim. **9** (1999), no. 4, 901–923 (electronic), Dedicated to John E. Dennis, Jr., on his 60th birthday.

[DR84] Philip J. Davis and Philip Rabinowitz, *Methods of numerical integration*, second ed., Computer Science and Applied Mathematics, Academic Press, Inc., Orlando, FL, 1984.

[Dun85] D. A. Dunavant, *High degree efficient symmetrical Gaussian quadrature rules for the triangle*, International Journal for Numerical Methods in Engineering **21** (1985), no. 6, 1129–1148.

[EG04] Alexandre Ern and Jean-Luc Guermond, *Theory and practice of finite elements*, Applied Mathematical Sciences, vol. 159, Springer-Verlag, New York, 2004.

[Fle70] R. Fletcher, *A new approach to variable metric algorithms*, The Computer Journal **13** (1970), no. 3, 317–322.

[Gol70] Donald Goldfarb, *A family of variable-metric methods derived by variational means*, Math. Comp. **24** (1970), 23–26.

[Gri80] J.D. Gribble, *Interpolatory inner product quadrature formulas*, BIT Numerical Mathematics **20** (1980), no. 4, 466–474.

[Gri82] J. D. Gribble, *Inner product quadrature formulas exact on maximal product spaces of functions*, J. Comput. Appl. Math. **8** (1982), no. 2, 73–79.

[GW69] Gene H. Golub and John H. Welsch, *Calculation of Gauss quadrature rules*, Math. Comp. **23** (1969), no. 106, loose microfiche suppl, A1–A10.

[Kno07] L. Knockaert, *A bilinear quadrature rule for the finite Hankel transform*, AFRICON 2007, Sept 2007, pp. 1–4.

[LZ87] Eberhard Luik and Karl Zeller, *Numerische approximation von bilinearformen*, Results in Mathematics **11** (1987), no. 3-4, 374–383.

[McG79] Joseph F. McGrath, *Gaussian product-type quadratures*, Applied Mathematics and Computation **5** (1979), no. 3, 265 – 280.

[OW95] Michael L. Overton and Robert S. Womersley, *Second derivatives for optimizing eigenvalues of symmetric matrices*, SIAM J. Matrix Anal. Appl. **16** (1995), no. 3, 697–718.

[RB14] Ernest K. Ryu and Stephen P. Boyd, *Extensions of Gauss quadrature via linear programming*, Foundations of Computational Mathematics (2014), 1–19.

[RMF77] W. C. Rheinboldt, C. K. Mesztenyi, and J. M. Fitzgerald, *On the evaluation of multivariate polynomials and their derivatives*, BIT Numerical Mathematics **17** (1977), no. 4, 437–457.

[SF95] Alexander Shapiro and Michael K. H. Fan, *On eigenvalue optimization*, SIAM J. Optim. **5** (1995), no. 3, 552–569.

[Sha70] D. F. Shanno, *Conditioning of quasi-Newton methods for function minimization*, Math. Comp. **24** (1970), 647–656.

[Str71] A. H. Stroud, *Approximate calculation of multiple integrals*, Prentice-Hall Inc., Englewood Cliffs, N.J., 1971, Prentice-Hall Series in Automatic Computation.

[VR14] B. Vioreanu and V. Rokhlin, *Spectra of multiplication operators as a numerical tool*, SIAM J. Sci. Comput. **36** (2014), no. 1, A267–A288.

[XG10] Hong Xiao and Zydrunas Gimbutas, *A numerical algorithm for the construction of efficient quadrature rules in two and higher dimensions*, Comput. Math. Appl. **59** (2010), no. 2, 663–676.

1.

**1. Report Type**

Final Report

**Primary Contact E-mail**
**Contact email if there is a problem with the report.**

strain@math.berkeley.edu

**Primary Contact Phone Number**
**Contact phone number if there is a problem with the report**

5106423656

**Organization / Institution name**

University of California

**Grant/Contract Title**
**The full title of the funded effort.**

FAST IMPLICIT METHODS FOR ELLIPTIC MOVING INTERFACE PROBLEMS

**Grant/Contract Number**
**AFOSR assigned control number. It must begin with "FA9550" or "F49620" or "FA2386".**

FA9550-11-1-0242

**Principal Investigator Name**
**The full name of the principal investigator on the grant or contract.**

John Strain

**Program Manager**
**The AFOSR Program Manager currently assigned to the award**

Dr. Jean-Luc Cambier

**Reporting Period Start Date**

09/30/2011

**Reporting Period End Date**

09/30/2015

**Abstract**

Two notable advances in numerical methods were supported by this grant.

First, a fast algorithm was derived, analyzed, and tested for the Fourier transform of piecewise polynomials given on d-dimensional simplices in D-dimensional Euclidean space. These transforms play a key role in computational problems ranging from medical imaging to partial differential equations, and existing algorithms are inaccurate and/or prohibitively slow for d > 0. The algorithm employs low-rank approximation by Taylor series organized in a butterfly scheme, with moments evaluated by a new dimensional recurrence and simplex quadrature rules. For moderate accuracy and problem size it runs orders

of magnitude faster than direct evaluation, and one to three orders of magnitude slower than the classical uniform Fast Fourier Transform.

Second, bilinear quadratures ---which numerically evaluate continuous bilinear maps, such as the L2 inner product, on continuous f and g belonging to known finite-dimensional function spaces---were analyzed and developed. Such maps arise in Galerkin methods for differential and integral equations. Bilinear quadratures were constructed over arbitrary D-dimensional domains. In one dimension, integration rules of this type include Gaussian quadrature for polynomials and the trapezoidal rule for trigonometric polynomials. A numerical procedure for constructing bilinear quadratures was developed and validated.

**Distribution Statement**

**This is block 12 on the SF298 form.**

Distribution A - Approved for Public Release

**Explanation for Distribution Statement**

**If this is not approved for public release, please provide a short explanation. E.g., contains proprietary information.**

**SF298 Form**

**Please attach your SF298 form. A blank SF298 can be found here. Please do not password protect or secure the PDF The maximum file size for an SF298 is 50MB.**

sf0298_final.pdf

**Upload the Report Document. File must be a PDF. Please do not password protect or secure the PDF . The maximum file size for the Report Document is 50MB.**

rep.pdf

**Upload a Report Document, if any. The maximum file size for the Report Document is 50MB.**

**Archival Publications (published) during reporting period:**

S Govindjee, T J Mitchell, J Strain and R L Taylor,
Convergence of an efficient local least-squares fitting method for bases with compact support. Comp. Meth. Appl. Mech. Engng. 213-216, 84-92 (2012).

**Changes in research objectives (if any):**

**Change in AFOSR Program Manager, if any:**

Dr. Fariba Fahroo -> Dr. Jean-Luc Cambier

**Extensions granted or milestones slipped, if any:**

1-year No-cost extension

**AFOSR LRIR Number**

**LRIR Title**

**Reporting Period**

**Laboratory Task Manager**

**Program Officer**

**Research Objectives**

**Technical Summary**

**Funding Summary by Cost Category (by FY, $K)**

| | Starting FY | FY+1 | FY+2 |
|---|---|---|---|
| Salary | | | |
| Equipment/Facilities | | | |
| Supplies | | | |
| Total | | | |

**Report Document**

**Report Document - Text Analysis**

**Report Document - Text Analysis**

**Appendix Documents**

## 2. Thank You

**E-mail user**

Dec 05, 2015 17:31:26 Success: Email Sent to: strain@math.berkeley.edu